



CISCO CONFIDENTIAL



SDK Developer's Guide for CiscoWorks Common Services 3.0.5

Americas Headquarters

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Customer Order Number:
Text Part Number: OL-xxxx-xx

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCSP, CCVP, the Cisco Square Bridge logo, Follow Me Browsing, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, and iQuick Study are service marks of Cisco Systems, Inc.; and Access Registrar, Aironet, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, FormShare, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, iQ Expertise, the iQ logo, iQ Net Readiness Scorecard, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, *Packet*, PIX, Post-Routing, Pre-Routing, ProConnect, RateMUX, ScriptShare, SlideCast, SMARTnet, The Fastest Way to Increase Your Internet Quotient, and TransPath are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0601R)

SDK Developer's Guide for CiscoWorks Common Services 3.0.5
Copyright © 2006, Cisco Systems, Inc. All rights reserved.



CISCO CONFIDENTIAL

CONTENTS

Preface 33

Audience	1-33
Conventions	1-33
Related Documentation	1-34
Document Organization	1-35
Obtaining Documentation	1-37
Cisco.com	1-37
Product Documentation DVD	1-37
Ordering Documentation	1-37
Documentation Feedback	1-37
Cisco Product Security Overview	1-38
Reporting Security Problems in Cisco Products	1-38
Product Alerts and Field Notices	1-39
Obtaining Technical Assistance	1-39
Cisco Technical Support & Documentation Website	1-39
Submitting a Service Request	1-40
Definitions of Service Request Severity	1-40
Obtaining Additional Publications and Information	1-41

About CWCS

CHAPTER 1

Introducing CWCS 1-1

CWCS Release Model	1-1
Benefits of Using CWCS	1-3
What's New in CWCS	1-3
What's New in This Guide	1-5
Understanding the CWCS Structure	1-5
Using CWCS Components	1-7
How CWCS is Distributed	1-9
Installation Interface Options	1-9
CD Image Structure	1-9
CD Image Structure for Windows	1-10
CD Image Structure for Solaris	1-10

CISCO CONFIDENTIAL

Third-Party Tools 1-10
 Where to Find the CWCS SDK 1-15
 For Further Assistance 1-15

CHAPTER 2

FAQs and Programming Hints 2-1

General Topics 2-1
 Daemon Manager 2-4
 Online Help 2-5

CHAPTER 3

Understanding the CWCS Directory Structure 3-1

About CWCS Directory Policies 3-1
 About the CWCS Top-Level Runtime Directories 3-1
 About the CWCS Common Directories 3-2
 About the CWCS Solaris-Specific Directories 3-3
 About the CWCS UNIX-Specific Directories 3-3
 About the CWCS Windows-Specific Directories 3-3
 About CWCS File Permissions 3-4
 About CWCS Property Files 3-4
 About CWCS Log Files 3-7

CHAPTER 4

Understanding the CWCS Execution Environment 4-1

Understanding the Java Application Launch Process 4-1
 Launching a Java Application 4-2
 Using Servlets and JSPs with Tomcat 4-5
 Using JavaBeans 4-5

CHAPTER 5

Getting Started with CWCS 5-1

How CWCS Works 5-1
 Installing CWCS 5-2
 Enabling CWCS Services 5-3
 Understanding CWCS Service Bundles 5-3
 Using CWCS Service Bundles 5-3
 Registering for CWCS Services 5-4
 Enabling New Service Bundles from the Command Line 5-5
 Using CMFEnable 5-5
 Interacting with CWCS 5-7
 Designing the User Interface 5-7

CISCO CONFIDENTIAL

Adding Your Application to the CiscoWorks Home Page	5-8
Using the Backend Services on the CWCS Server	5-8
Using the CWCS Support Tools	5-9
Getting Up to Speed Quickly	5-9

About CWCS Shared Services**CHAPTER 6****Using Shared Services 6-1**

Understanding Shared Services	6-1
About the Shared Services Components	6-4
About the CiscoWorks Home Page Component	6-5
About Web Server and Servlet Engine Components	6-6
About the Cisco Management Integration Center (CMIC) Component	6-6
About the Security System Components	6-7
About the Database Components	6-8
About the Backup and Restore Components	6-8
About the Device List and Credentials Repository (DCR) Components	6-9
About the Core Client Registry (CCR) Component	6-10
About the Core Logging Component	6-11
About the Online Help Component	6-12
About the Daemon Manager Component	6-12
About the Job and Resource Manager (JRM) Component	6-13
About the Event Services Software (ESS) Component	6-13
About the Event Distribution System (EDS) Component	6-13
About the Installation Framework Component	6-14
About the Java Plug-in Component	6-14
About Diagnostic and Support Components	6-14
About SNMP Service Components	6-15
About NT Service Components	6-15
About Device Center Components	6-15

CHAPTER 7**Using the CiscoWorks Home Page 7-1**

Understanding CWHP	7-1
About the CWHP Interface	7-2
How CWHP Works	7-3
How CWHP Uses CMIC	7-4
How CWHP Handles Security	7-5
About the CWHP Runtime Structure	7-5
Integrating Your Application with CWHP	7-6

CISCO CONFIDENTIAL

Registering Your UII-based Application with CWHP 7-6

- Implementing CWHP Security 7-7
- Implementing Special License Checks 7-8
- Handling CWHP Messages 7-8
- Migrating to CWHP 7-9

CHAPTER 8

Using Web Servers and Servlet Engines 8-1

- Understanding the CWCS Web Server and Servlet Engine 8-1
- About the CWCS Web Server and SSL 8-2
- Using CWCS Web Servers and Servlet Engines 8-2
 - About the JRE Version 8-2
 - About Apache Version and Access Control 8-2
- Servlet Engines and Runtime Directory 8-3
 - Runtime Structure for New Components 8-3
 - Existing Tomcat Based Components/Applications 8-4
 - New Tomcat Based Components/Applications 8-4
 - Runtime Structure for CiscoWorks Common Services Webapps 8-5
 - Runtime Structure for DCR 8-6
 - Runtime Structure for CMIC 8-6
 - Runtime Structure for Device Center 8-6
- Implications of HTML Based Login 8-7

CHAPTER 9

Integrating Applications with CMIC 9-1

- Understanding CMIC 9-1
- Using CMIC Services 9-2
 - Registering Applications 9-2
 - Unregistering Applications Through Command Line 9-3
 - Registering Applications Through Command Line 9-3
 - Querying an Application 9-4
 - Calling an Application 9-4
 - Integrating CMIC with CWHP, Device Center, and Setup Center 9-4
 - Wrapper Java Code 9-5
 - System Flow for CWHP using CMIC 9-6
 - About the CMIC APIs 9-7
 - About the Management Service Template 9-7
 - Component Interaction 9-8
 - About CMIC Registry Dependencies 9-9
 - Sample MST File 9-9

CISCO CONFIDENTIAL**CHAPTER 10****Using the Security System 10-1**

Understanding CWCS Security	10-1
About Client-to-Server Security	10-2
About Server Internal Security	10-2
User Name Length Restrictions	10-3
Using CWCS Single Sign-On	10-3
How the Login Protocol Works	10-4
How the Logout Protocol Works	10-5
About Server-Imposed Security	10-5
About Administrator-Imposed Security	10-6
About Server-to-Device Security	10-6
About Secure Shell (SSH)	10-6
Using CWCS Server Security	10-6
About General Security	10-7
Application Integration with CAM	10-7
How CAM Cache works	10-8
API Level Details	10-9
Authorization Checking	10-9
Setting Up Server Internal Security	10-10
Using the Shared Secret Client API	10-10
Client Side API Details	10-11
SecretClient.secretLogon	10-12
SecretClient.getErrCode	10-12
SecretClient.getErrReason	10-13
SecretClient.doPost	10-13
SecretClient.dumpResponse	10-14
Setting Up Server-to-Device Security	10-14
Integrating a New Application	10-16
Securing Applications	10-17
Securing Java Servlets	10-17
Securing Java Applets	10-17
Backend Perl Script	10-17
Java Server Pages (JSP)	10-18
Creating Auto Login Pages	10-18
Performing Encryption	10-19
Handling Symmetrical Encryption	10-19
Handling Asymmetrical (One-Way) Encryption	10-20
Stopping Eavesdropping Using SSL	10-20
Why Use SSL in CWCS?	10-20

CISCO CONFIDENTIAL

SSL Support in CWCS 10-21
 What Kind of SSL Support is Available in CWCS? 10-21
 SSL-Enabling Your Application 10-21
 Configuring System Identity Setups 10-21
 Configuring a Cisco.com User and Password 10-22

CHAPTER 11

Using the Database APIs 11-1

Understanding the CWCS Database 11-2
 What's New in This Release 11-2
 Understanding the Tools 11-2
 Database Access Methods 11-3
 Types of Database Servers 11-3
 JDBC Access Methods 11-4
 ODBC Access Methods 11-4
 Perl Access Methods 11-4
 Connection Strings 11-5
 Understanding the NMTG Database Delivery Process 11-5
 Setting Up a New Database 11-6
 Creating the ODBC Database Definition File 11-7
 Creating the Database Template File 11-7
 Creating the odbc.tmplorig Template File 11-7
 Customizing the odbc.tmpl File 11-8
 Enabling Database Password Encryption 11-9
 Creating the Backup Manifest Files 11-10
 Creating the Database Backup Manifest File 11-11
 Creating the Application Backup Manifest File 11-11
 About the Database Property Files and Settings 11-12
 About the Database Server Property File 11-12
 About Private Property Files 11-13
 Managing the Database Engine 11-13
 Understanding Port IDs 11-14
 Creating a Database Port 11-15
 Changing the Database Port 11-16
 Dynamically Allocating a Port ID 11-17
 Performing a Quick Integration 11-17
 Using the Sybase Database 11-18
 Before You Begin 11-19
 Setting Up Your Environment 11-19
 Initializing a New Database 11-19

CISCO CONFIDENTIAL

Creating a New Database	11-20
Step 1: Change the User ID and Password	11-21
Step 2: Create and Populate DbVersion and DbVersionHistory	11-21
Step 3: Install the Database Files	11-23
Updating the Database Password	11-24
Starting and Stopping Database Engines	11-25
Starting a Database Engine	11-25
Stopping a Database Engine	11-27
Creating and Closing Database Connections	11-28
Connecting to a Database	11-28
Closing a Database Connection	11-30
Examining the Contents of a Database	11-31
Creating a DSN	11-31
Accessing Your Data	11-32
Backing Up Your Database	11-32
Debugging and Troubleshooting the Database	11-33
Managing Database Log Files	11-33
Ensuring Sufficient Temporary Space	11-33
Optimizing Query Processing	11-33
Verifying a Database	11-34
Reinitializing a Database	11-35
Cleaning Up Other Application Files	11-36
Database API Command Reference	11-37
Enabling the CWCS Database Engine	11-37
Compiling and Running a Database	11-37
Code Samples	11-38
Using Java to Read a Database	11-38
Using ODBC to Access a Table	11-40
Using Perl to Access a Database	11-40
Using JDBC API Wrappers	11-41
DBClient	11-41
DBResult	11-42
Class DBUtil	11-43
DBConnection	11-44
Using CWCS Perl APIs	11-46
Programming Tips for Perl APIs	11-46
Perl API Summaries	11-46
addManifestFiles	11-47
check_create	11-48
checkDb	11-48

CISCO CONFIDENTIAL

deleteDbVersionData 11-48
 deleteManifestFiles 11-49
 getDbVersionData 11-49
 getManifestFiles 11-50
 setDbVersionData 11-51
 StartDb 11-51
 StopDb 11-51
 unloadDbVersionData 11-52
 Using the Database Utilities 11-52
 configureDb.pl 11-53
 dbinit 11-54
 dbMonitor 11-55
 DBPing 11-56
 dbpasswd.pl 11-57
 dbreader.pl 11-59
 dbRestoreOrig 11-59
 dbvalid 11-60
 runsql 11-60

CHAPTER 12

Using Backup and Restore 12-1

Using CWCS Backup 12-1
 CWCS Backup 12-1
 How CWCS Backups are Stored 12-2
 Running CWCS Backups 12-3
 Offline Backup 12-4
 Using CWCS Restore 12-4
 CWCS Restore: Changes for CWCS 3.0 12-4
 Understanding the CWCS Restore Framework 12-5
 Running CWCS Restores 12-6
 Guidelines for Writing CWCS Restore Application Adaptors 12-6
 Sample CWCS Restore Application Adaptor 12-8
 CWCS Backup and Restore API Command Reference 12-9
 backup.pl 12-10
 restorebackup.pl 12-11
 copyFileToNMSROOT 12-12
 copyFolderToNMSROOT 12-12
 getCMFVersion 12-13
 getCMFPatchVersion 12-13
 getNMSROOT 12-14

CISCO CONFIDENTIAL

getArchiveNMSROOT	12-14
getFolderSeperator	12-14
getLogFileName	12-15
getTempFolder	12-15
isWindows	12-15
redirectToScreen	12-16
redirectToLog	12-17
restoreDatabase	12-17
StandardDbRebuild	12-18
Restoring a Corrupt Database	12-18
Restoring a Corrupt Database from a Previous Backup	12-19
Recovering Part of a Corrupt Database	12-19
Recovering Part of a Corrupt Database On Windows Platforms	12-19
Recovering Part of a Corrupt Database On Solaris Platforms	12-20
Abandoning a Corrupt Database	12-21

CHAPTER 13

Using the Core Client Registry	13-23
Understanding CCR	13-23
About the CCR Components	13-24
CCR Local System Data (LSD) Component	13-24
CCRProcess Component	13-26
CCRInterface Component	13-26
CCREntry Component	13-26
CCRResponse Component	13-26
About CCR System Flow	13-27
Adding an LSD Entry (Installation)	13-27
Removing an LSD Entry (Uninstall)	13-27
Modifying an LSD Entry (Patching/Upgrading)	13-27
Retrieving an LSD Entry	13-28
About CCR Data Structures	13-28
Local System Data (LSD) Data Structure	13-28
CCREntry Data Structure	13-31
CCRResponse Data Structure	13-31
Using the CCR C++ API	13-31
CCRInterface Functions	13-31
CCREntry Functions	13-36
CCRResponse Functions	13-43
Using the CCR API: Example	13-46
Using the CCRAccess Client	13-47

CISCO CONFIDENTIAL

- Scripting CCRAccess 13-49
- Using the CCRAccess DLL 13-50
- Using the CCR Java Interface 13-51
- Encrypting and Decrypting CCREntry Values 13-52
 - Encrypting Entry Names 13-52
 - Decrypting Entry Names 13-53
 - Encrypting Entry Data 13-54
 - Decrypting Entry Data 13-55
 - Encrypting Entry Locations 13-55
 - Decrypting Entry Locations 13-56
 - Encrypting Custom Entries 13-57
 - Decrypting Custom Entries 13-58

CHAPTER 14

Using the Device Credentials Repository 14-1

- Understanding DCR 14-2
 - About DCR Features and Benefits 14-2
 - How DCR Works 14-3
 - About the DCR Modes 14-4
 - About the DCR Components 14-5
 - How DCR Masters and Slaves Interact 14-6
 - How DCR Adds Devices 14-7
 - How DCR Modifies Devices 14-7
 - How DCR Deletes Devices 14-8
 - How DCR Secures Device and Credentials Data 14-8
 - About DCR Data Storage 14-9
 - About the DCR Device ID 14-9
 - How DCR Stores Attributes 14-10
 - How DCR Stores Credentials 14-10
 - How DCR Stores Proxy Device Data 14-11
 - How DCR Stores Enable-Mode Passwords 14-13
 - About User-Defined Fields 14-13
 - Integrating DCR with OGS 14-13
 - Integrating DCR with ACS 14-14
 - About DCR Events 14-15
 - About the DCR Domain ID and Transaction ID 14-16
 - About DCR Device Events 14-17
 - About DCR Process Events 14-19
 - About DCR Restore Events 14-21
 - About DCR Events During Backup and Restore 14-22

CISCO CONFIDENTIAL

Using DCR	14-25
Getting Started With DCR	14-26
DCR Tasks to Perform During Application Startup	14-27
Using the DCR APIs	14-27
About the DCR APIs	14-28
Creating the DCRProxy Object	14-29
Creating the APIExtraInfo Object	14-30
Adding Devices to DCR	14-31
Updating a DCR Device	14-33
Adding and Updating Devices in Bulk	14-33
Retrieving DCR Device Objects	14-34
Retrieving DCR Devices in Bulk	14-35
Retrieving Data From a Device Object	14-36
Comparing Two Device Objects	14-37
Registering Third-Party Applications with DCR	14-37
Guidelines for DCR Application Development	14-38
DCR Error Codes and Interpretations	14-38
Responding to DCR Events	14-40
Using DCR Domain and Transaction IDs	14-44
Using the DCR Command-Line Interface	14-46
Enhancing DCR Performance	14-49

CHAPTER 15

Using the Core Logging API	15-1
About the Core Logging API Structure	15-1
Using the Core Logging API	15-2
Initializing the Core Logging API	15-2
Creating Debug Messages	15-2
Creating Information Messages	15-2
Creating Warning Messages	15-3
Creating Error Messages	15-3
Creating Fatal Messages	15-3
Creating Auditing Messages	15-3
Altering Priority for Category	15-3
Adding Logging Location to CCR	15-4
Adding Logging Category and Priority to CCR	15-4
About the Core Logging API Interface Design	15-4
About the Logger Interface	15-4
About the JavaLogger Interface	15-6
About the Auditlog Interface	15-8

CISCO CONFIDENTIAL**CHAPTER 16****Adding Online Help 16-1**

- Overview of Online Help 16-2
 - How Help Is Displayed 16-2
 - Understanding the Help Engine 16-4
 - Displaying Help Topics 16-4
 - Linking Context-Sensitive Help Buttons 16-4
 - Creating the Main Help Contents and Index 16-5
 - Handling Error Conditions 16-5
 - Summarizing the Display Process 16-5
 - Understanding the Search Engine 16-6
 - Displaying the Search Dialog Box 16-7
 - Searching the Files and Displaying the Search Results 16-7
 - Summarizing the Search Process 16-8
 - Understanding Mapping Files 16-9
 - Mapping File Conventions and Requirements 16-10
 - Mapping File Line Types 16-10
 - Sample Mapping File 16-12
- Implementing Help: Engineering Tasks 16-13
 - Installing the Help Packages 16-13
 - Adding a Call to the Help Engine 16-14
 - Calling Help From a Java Application 16-14
 - Calling Help From an HTML-Based Application 16-15
 - Updating the Mapping File 16-15
 - Packaging the Help Files 16-16
- Implementing Help: Writing Tasks 16-18
 - Selecting an Authoring Tool 16-19
 - Setting Up Your Authoring Environment 16-19
 - Setting Up the Native HTML Authoring Environment 16-19
 - Setting Up the XML Authoring Environment 16-20
 - Setting Up the FrameMaker/WebWorks Authoring Environment 16-21
 - Creating the Help Topic Files 16-22
 - Maintaining Your Help System's Mapping File 16-23
 - Creating the Mapping File 16-23
 - Defining the Main Help Page Contents and Index 16-24
 - Adding Search Support 16-27
 - Maintaining the Search Index File 16-28
 - Delivering Your Help System 16-28
- Adding Drop-In Help Systems 16-29

CISCO CONFIDENTIAL**CHAPTER 17****Using the Daemon Manager 17-1**

Understanding the Daemon Manager	17-1
Using the Daemon Manager	17-2
Starting and Stopping the Daemon Manager	17-2
Using the Daemon Manager Command Line Interface	17-3
Using the Daemon Manager Application Programming Interface	17-3
Using the Daemon Manager C++ Interface	17-4
Using the Daemon Manager Java Interface	17-4
Using a Ready File to Ensure Process Dependencies are Met	17-4
Writing Messages to Log Files	17-5
Daemon Manager Command Reference	17-5
Daemon Manager Command Line Utilities	17-6
pdexec	17-6
pdrapp	17-6
pdreg	17-7
pdshow	17-9
pdterm	17-10
Daemon Manager ANSI C and C++ Commands	17-11
dMgtClose	17-11
dMgtCreateReadyFile	17-12
dMgtErr	17-12
dMgtGetMsg	17-13
dMgtInit	17-13
dMgtIsShutdown	17-14
dMgtProcessMsg	17-14
dMgtSendStatus	17-14
GetConFile	17-15
GetDescriptor	17-15
GetDmgtHostAndPort	17-16
ValidatePgmPath	17-16
Daemon Manager Java Methods	17-17
CreateReadyFile	17-18
GetCmdType	17-18
GetDescriptor	17-18
GetErr	17-19
GetMsg	17-19
GetServerInfo	17-19
GetStatusMsg	17-20
IsShutdownRequest	17-20

CISCO CONFIDENTIAL

- ProcessMsg 17-20
- ReqStatus 17-21
- SendBusyMsg 17-21
- SendErrMsg 17-21
- SendOkMsg 17-22
- StartProcess 17-22
- StopProcess 17-22
- Status 17-23

CHAPTER 18

Using the Job and Resource Manager 18-1

- Understanding JRM Services 18-2
 - Managing JRM Services 18-3
 - Scheduling Jobs 18-3
 - Locking Resources 18-4
 - Locking Resources from Another Application 18-4
 - Locking Parts of a Device 18-5
- Understanding the JRM Architecture 18-5
 - An Overview of the JRM Architecture 18-5
 - Understanding the JRM Server 18-7
 - About Jobs and Resources 18-7
 - About JRM Server Classes 18-8
 - About the IDL Interface 18-8
 - About the Helper API 18-9
 - About JRM Events 18-9
 - Understanding the Job Browser 18-10
 - How JRM Relates to Other CWCS Components 18-11
- Enabling JRM 18-12
- Using JRM from a Java Application 18-12
 - Establishing a Connection 18-12
 - Creating a Job 18-14
 - Setting the Job Status 18-15
 - Getting Job Descriptions 18-16
 - Handling an Unapproved Job 18-16
 - Enabling a Disabled Job 18-17
 - Handling a Crashed Job 18-18
 - Locking and Unlocking a Device 18-19
 - Handling an Unavailable Resource 18-19
 - Accessing a Locked Device 18-20
- Using JRM from a Web Browser 18-21

CISCO CONFIDENTIAL

Customizing the Job Browser Button Behaviors	18-22
Using JRM from the Command Line	18-24
Job Command Line Interface	18-25
Lock Command Line Interface	18-25
JRM Command Reference	18-26
About the Job and Resource Lock Attributes	18-26
About Displayed Job Status Values	18-28
About the Job Manager Methods	18-29
job_cancel	18-31
job_cancel_instance	18-31
job_cancel_event	18-32
job_cancel_instance_event	18-32
job_create	18-33
job_create_hist	18-33
job_delete	18-34
job_delete_instance	18-34
job_enum	18-35
job_enum_hist	18-35
job_get_info	18-36
job_get_info_hist	18-36
job_get_result	18-37
job_get_schedule	18-37
job_get_schedule_string	18-38
job_run	18-38
job_set_approved	18-39
job_set_info	18-39
job_set_info_hist	18-40
job_set_progress_string	18-40
job_set_reference	18-40
job_set_result	18-41
job_set_resume	18-41
job_set_schedule	18-42
next	18-42
next_n	18-43
release	18-43
About the Lock Manager Methods	18-44
enum_job_locks	18-44
find_lock	18-45
get_lock	18-45
lock	18-46

CISCO CONFIDENTIAL

- lock_n 18-46
- next 18-47
- next_n 18-47
- release 18-48
- unlock 18-48
- unlock_job 18-49
- unlock_n 18-49
- About the Helper API Methods 18-49
 - get_job_id 18-51
 - get_job_instance_id 18-51
 - get_job_info 18-52
 - get_job_info_hist 18-52
 - get_lock_info 18-52
 - getOrbConnectionProperties 18-53
 - getScheduleString 18-53
 - getStateStrings 18-53
 - is_server_running 18-54
 - lock 18-54
 - lock_n 18-55
 - set_completion_state 18-55
 - set_progress 18-55
 - unlock 18-56
 - unlock_all 18-56
- About the JRM Java Constants 18-56
 - Parsing ESS Messages 18-58
- Using the Job Command-Line Commands 18-59
 - approve 18-59
 - cancel 18-60
 - create 18-60
 - delay 18-60
 - delete 18-61
 - getnextschedule 18-61
 - reject 18-61
 - resume 18-61
 - run 18-62
 - schedule 18-62
 - suspend 18-62

CHAPTER 19	Using Event Services Software	19-1
	Understanding ESS Subsystems	19-1

CISCO CONFIDENTIAL

How Does ESS Work?	19-2
How Is ESS Organized?	19-2
Using Tibco's Rendezvous	19-2
About Subject Names and Event Formats	19-3
How Subject Names are Structured	19-3
Choosing Subject Names and Namespaces	19-3
Subscribing with Wildcards	19-5
About ESS Event Formats	19-5
About Reserved Subject Names	19-5
Support for Map Messages	19-6
Using the Lightweight Messaging Service	19-8
Understanding LWMS	19-9
About the LWMS Components	19-9
How LWMS Works	19-10
About LWMS Message Queues	19-12
About JMS API Support	19-12
About LWMS Server Logging	19-12
About LWMS Usage Assumptions	19-12
About Tibco-LWMS Gateway Support	19-13
Configuring LWMS	19-13
Configuring Client Properties	19-13
Configuring Server Properties	19-14
Using the LWMS API	19-15
Creating a Mailbox with LWMS	19-15
Posting a Message to a Mailbox with LWMS	19-16
Polling Mailboxes for New Messages with LWMS	19-16
Removing a Message Listener with LWMS	19-16
Filtering Messages with LWMS	19-16
Using the JMS API	19-18
Creating a Mailbox with JMS APIs	19-18
Posting a Message to a Mailbox with JMS	19-18
Polling Mailboxes for New Messages with JMS	19-18
Removing a Message Listener with JMS	19-18
Using JMS Message Selectors	19-19
LWMS Command Reference	19-19
LWMS Native API Messaging Methods	19-19
JMS to LWMS Mappings	19-21

CISCO CONFIDENTIAL

CHAPTER 20

Using the Event Distribution System 20-1

- About the EDS Components 20-1
 - About the EDS Event Server 20-2
 - About the EDS Event Message 20-2
 - About the EDS Atom Service 20-2
 - About the EDS Manager 20-2
 - About the EDS Class Loader 20-3
 - About the EDS New Event Message Fields 20-3
 - About the EDS Event Logger 20-3
 - About the EDS Event Logger Display 20-3
 - About the EDS Named Event Filter Service 20-3
 - About the EDS Event to Trap Converter 20-3
 - About the EDS Trap to Event Converter 20-3
- Using the EDS Programmatic Interface 20-4
 - About EDS Events 20-4
 - Formatting EDS Events 20-5
 - Defining and Registering EDS Event Atoms 20-5
 - Using the EDS Atom Definition File 20-6
 - Using the Atom Service Executables 20-8
 - Using the EDS Java Interface Classes 20-8
 - Registering EDS Application Events 20-10
 - Using the EDS Trap to Event Service 20-10
 - Using the EDS Trap Receiver Framework 20-10
 - Using the Trap Receptor 20-11
 - Using the Trap Receiver Configuration File 20-12
 - Using TrapInclude/TrapExclude Statements 20-12
 - Creating Trap Actions 20-13
 - Matching Trap Records 20-14
 - Using the TrapToEDS Converter 20-14
 - How the TrapToEDS Conversion Table is Used 20-14
 - Using the TrapLaunch Action 20-15
 - Using the TrapEcho Action 20-15
 - Setting Trap Receiver Properties 20-16
 - Using the Generic Consumer Framework 20-16
 - Using the GCF Configuration File 20-16
 - Using the GCF Admin Display 20-17
 - Creating Generic Consumers 20-17
 - Using the Event to Trap Converter with Generic Consumers 20-17
- Using EDS to Publish Events 20-18

CISCO CONFIDENTIAL

About the EDS-Published Event Types	20-19
About the EDS-Published Severity Codes	20-19
Registering Your Application with EDS	20-20

CHAPTER 21

Using the Installation Framework	21-1
About the Installation Framework	21-1
What's New in This Release	21-2
Understanding the CWCS Installation Framework	21-2
Understanding Installation Team Responsibilities	21-3
Understanding Developer Responsibilities	21-3
Getting Started with the Installation Framework	21-4
Third-Party Tools for Installation Framework	21-4
Understanding Install Component and Image Structures	21-4
Building an Installable Image	21-5
Selecting Package Names	21-6
Specifying Package Properties	21-6
Understanding the Package Properties File	21-7
Understanding Suite Properties	21-10
Creating the Table of Contents	21-11
How the Installer Processes Properties	21-18
Specifying Properties	21-19
Preparing Installation Protopackages	21-19
Including Files in the Protopackage	21-20
Using the Installation Framework	21-21
Understanding the Common Services Upgrade	21-21
Understanding and Implementing the casuser	21-21
Providing Licensing Information During Installation	21-22
Installing Database Upgrades	21-22
Upgrade Installation Paths and Strategies	21-22
About the CWCS Upgrade Mechanism	21-23
Adding Unauthenticated URLs	21-26
Overriding the Dependency Handler	21-27
Handling Patches	21-27
Patch Policy	21-27
Creating a Patch	21-27
Example: Making a Patch CD	21-28
Application Registration with ACS during Installation	21-31
Windows Installation Reference	21-32
Setting File Permissions During Installation on Windows	21-32

CISCO CONFIDENTIAL

- Writing Windows Scripts **21-33**
 - Using the Windows Installation Hooks **21-33**
 - Using the *pkg.rul* Installation File **21-34**
 - Using Installer Global Variables **21-35**
 - Preloading the Global List, IAnswerFile **21-35**
 - Reducing Windows Installation Time **21-36**
- Using the Windows Installation APIs **21-37**
 - Accessing and Setting Package Properties to Perform Version Comparisons **21-37**
 - Controlling Responses to Terminated Installations **21-45**
 - Processing Name=Value Pairs **21-46**
 - Sending Informational Messages to a Log File **21-48**
 - Informing the Installer That a Component Requires More Space **21-50**
 - Registering and Unregistering CWCS Daemons **21-51**
 - Running Commands in a Shell **21-52**
 - Locating the Root Directory Path Name **21-54**
 - Registering and Controlling Windows Services **21-55**
 - Using Generic Utilities **21-58**
 - Managing Passwords **21-62**
 - Configuring Tomcat **21-67**
 - Controlling Reboots **21-69**
- Using Windows Build Tools **21-70**
 - Step 1: Install Third-Party Tools for Windows **21-70**
 - Step 2: Install the Framework on Windows Platforms **21-70**
 - Step 3: Prepare the Make Image on Windows Platforms **21-71**
 - Debugging on Windows Platforms **21-72**
 - Example: Using Windows Build Tools **21-72**
- Customizing the Installation Workflow for Windows **21-76**
 - About the Installer Workflow **21-77**
 - Getting Started with Windows Installer Tools **21-77**
 - Creating the Installation Project File **21-77**
 - Creating Install Actions **21-78**
 - Creating Install Panels **21-79**
 - Specifying Conditions For Install Actions and Panels **21-82**
 - Creating the Install Staging Area **21-82**
 - Example: Adding Message Boxes to an Installation **21-83**
 - Example: Creating Custom Password Dialogs **21-84**
 - Example: Adding User Data to Show Details **21-85**
- Solaris Installation Reference **21-86**
 - Setting Ownership for Package Files on Solaris **21-86**
 - Setting Ownership from *package_name.owner* File **21-87**

CISCO CONFIDENTIAL

Understanding the <i>package_name.owner</i> File	21-88
Setting Ownership During Build/Installation	21-88
Setting Ownership Assignment Details	21-88
Creating the Answer File	21-89
Writing Solaris Scripts	21-90
Using the Solaris Installation Hooks	21-90
Where to Find Solaris Installation Examples	21-91
Using the Solaris Installation APIs	21-91
Using the Solaris Input/Output APIs	21-92
Using the Solaris Package APIs	21-97
Using the Solaris System APIs	21-100
Using the Solaris Installable Unit APIs	21-102
Using Solaris Build Tools	21-103
Step 1: Install Third-Party Tools On Solaris	21-103
Step 2: Install the Framework On Solaris Platforms	21-103
Step 3: Prepare the Make Image on Solaris	21-103
Customizing the Installation Workflow on Solaris	21-104
Debugging on Solaris	21-105
Verifying Packages on Solaris	21-105
Solaris Getting Started Example	21-106

CHAPTER 22**Using the Java Plug-in 22-1**

About the Java Plug-in Requirements	22-1
Using the Java Plug-in API	22-2
Accessing the JPI Configuration from CCR	22-2
Using Tags Java Plug-in	22-3
Using Client Local Resources	22-3
JPI Technology References	22-4

CHAPTER 23**Using the Diagnostic and Support Utilities 23-1**

Using Collect Server Info	23-1
What Data Does Collect Server Info Gather?	23-1
Customizing Collect Server Info	23-2
Running CollectServerInfo	23-2
Using the MDC Support Utility	23-3
About the MDC Support Utility Requirements	23-3
What Data Does the MDC Utility Collect?	23-3
Registering Alternative MDC Support Utilities	23-4
Running MDC Support	23-5

CISCO CONFIDENTIAL

- Using SNMP Set and Walk 23-6
 - About the SNMP Set and Walk Requirements 23-6
 - Running SNMP Set and Walk 23-6
 - Updating the MIBs for SNMP Walk 23-8
- Using Packet Capture 23-8
 - About the Packet Capture Utility Requirements 23-8
 - Running Packet Capture 23-9
- Using Logrot 23-10
 - Configuring Logrot 23-10
 - Running Logrot 23-11
 - Using Logrot Command Line Switches 23-12
 - Troubleshooting Logrot 23-12
 - Verifying Files and Time Cycles 23-12
 - Verifying Scheduled Tasks 23-13
 - Viewing the Scheduled Jobs Log File 23-13
 - Verifying Logrot Status 23-13
 - Known Problems with Logrot 23-13

CHAPTER 24

- Using SNMP Services 24-1**
 - Why SNMPv3? 24-1
 - How SNMP Support Works 24-2
 - Using CWCS SNMP Services 24-3
 - About the SNMP Classes in the Main Library 24-4**
 - About the SNMP Classes in the Futureapi 24-5

CHAPTER 25

- Using NT Services 25-1**
 - Understanding CWCS NT Services 25-1
 - About the NT TFTP Service 25-2
 - About the NT Telnet Service 25-2
 - About the NT Service APIs 25-3
 - About the NT RCP Service 25-3
 - About the CRMLogger Service 25-5
 - Using CWCS NT Services 25-6
 - Registering and Controlling NT Services 25-6
 - Writing Messages to Log Files 25-10

CHAPTER 26

- Using Device Center 26-1**
 - Understanding Device Center 26-1

CISCO CONFIDENTIAL

What You Can Do With Device Center	26-2
About Device Center Launch Points	26-2
What's Inside Device Center	26-2
About Device Center Dependencies	26-3
About the Device Center Runtime Structure	26-4
About the Device Center User Experience	26-4
Using Device Center With Your Application	26-5
Launching Device Center	26-5
Registering Your Application With Device Center	26-6
About the Device Center MST	26-7
Sample Device Center MST	26-10
MST XML-Schema	26-11
Creating and Registering the MST With CMIC	26-13
About Device Center Integration Tags	26-14
About UI Rendering Module	26-14
Providing Summary Information	26-15
Understanding PIDM	26-17
Bypassing PIDM Checks	26-17

CHAPTER 27**Using Product Instance Device Mapping 27-1**

Using the PIDM APIs	27-1
Creating the ProductToDeviceMapProxy Object	27-2
Mapping a Device or Marking a Device(s) as Managed	27-2
Unmapping a Device or Marking a Device(s) as Not Managed	27-2
Retrieving PIDM Information	27-2
Using the PIDM North-bound APIs	27-3
PIDM North-bound APIs	27-3
PIDM NB APIs and Associated Tasks	27-3
Creating the APIExtraInfo Object	27-4
Creating the ProductToDeviceMapNBProxy Object	27-4
Mapping a Device or Marking a Device(s) as Managed	27-4
Unmapping a Device or Marking a Device(s) as Not Managed	27-5
Retrieving PIDM Information	27-5

CHAPTER 28**Integrating Applications With Device Selector 28-1**

UI Integration	28-2
Integration with Search feature	28-2
Configuring Property files	28-3
Integration with Advanced Search Feature	28-3

CISCO CONFIDENTIAL

- Integration with Tree Generator 28-5
 - Tree Generator Changes for Device Selector Nodes 28-5
 - Tree Generator Changes for Search Implementations 28-5
 - Tree Generator Changes for Group Customization and Group Ordering 28-5

About CWCS Per-Product Services

CHAPTER 29

Using Per-Product Services 29-1

- Understanding Per-Product Services 29-1
- About the Per-Product Services Components 29-3
 - About the Object Grouping Service (OGS) Components 29-4
 - About the Common Services Transport Mechanism (CSTM) Components 29-4
 - About the Package Support Updater (PSU) Components 29-5
 - About the Common Incremental Device Support (CIDS) Component 29-5
 - About CWCS Licensing 29-6
 - About the User Interface Infrastructure 29-7

CHAPTER 30

Using Object Grouping Services 30-1

- Understanding OGS 30-2
 - About the OGS Components 30-2
 - Basic OGS Concepts 30-2
 - About OGS Groups 30-3
 - About OGS Group Types 30-4
 - About OGS Container Groups 30-5
 - About OGS Group Hierarchy 30-5
 - How Rules Are Constructed 30-6
 - Choosing to Implement OGS 30-6
- Implementing OGS Servers 30-7
 - Getting Started with OGS Server 30-7
 - How OGS Server Works 30-8
 - Using the OGS Server APIs 30-8
 - Customizing OGS Server Interfaces 30-11
 - Creating a Custom OGS Event Processor 30-15
 - Handling OGS Exceptions 30-16
- Creating OGS ASAs 30-17
 - Understanding ASA Infrastructure Modules 30-19
 - About the Rule Validator 30-20
 - About the Generic Schema 30-20
 - About the Rule Evaluator 30-21

CISCO CONFIDENTIAL

About the Mapping Schema	30-21
About the Rule Converter	30-21
About Node Rule Expressions	30-21
About Composite Rule Expressions	30-22
About the ASA Change Alerter	30-22
Customizing ASA Infrastructure Modules	30-22
Customizing the Rule Validator	30-23
Customizing the Generic Schema	30-23
Customizing the Rule Evaluator	30-26
Customizing the Mapping Schema	30-27
Customizing the Rule Converter and Rule Expressions	30-36
Customizing the ASA Change Alerter	30-36
Running a Customized ASA	30-36
Registering the ASA with OGS	30-36
Creating the ASA Configuration File	30-38
Example: Using the Generic SQL ASA	30-40
Creating an OGS GUI	30-40
Using OGS Secure Views	30-43
How Secure Views Work	30-43
Implementing Secure Views	30-45
Installing Secure Views	30-45
Using OGS SecurityContext	30-45
Using Secure Views With DCR IDs	30-46
Using Secure Views with Object Selector	30-46
Using Secure Views With the OGS Administrative GUI	30-47
Customizing Your Secure Views Implementation	30-47
Specifying a Non-Default Implementation	30-48
Using Secure Views Without DCR IDs	30-48
Using OGS Common and Shared Groups	30-49
Configuring OGSServer.properties	30-49
Configuring SharedGroups.properties	30-49
Implementing the SharedGroupObjectMapperlf Interface	30-50
OGS Utility Class for Common and Shared Groups	30-50
Using OGS 1.3 Client Side Enhancements	30-51
About the Enhanced OGS 1.3 Classes and Data Structures	30-51
Controlling the Display of Wizard Steps	30-53
Integrating OGS 1.3 With Your Application	30-53
Using OGS 1.4 Enhancements	30-58
Integrating OGS 1.4 With Your Application	30-59

CISCO CONFIDENTIAL

Integrating Configurable Display Name for Class Names Feature with Applications 30-67

CHAPTER 31

Using the Common Services Transport Mechanism 31-1

- Understanding CSTM 31-1
- Installing CSTM 31-2
 - Installing Basic CSTM 31-2
 - Installing CSTM with the Tomcat Servlet Engine 31-3
- Controlling CSTM Logging 31-3
 - Setting Up CSTM Logging 31-4
 - Setting the CSTM Logging Levels 31-4
 - Changing the CSTM Logging Destination 31-4
 - Starting a Log4j Server 31-5
 - Viewing the CSTM Log File 31-5
- Publishing Objects 31-6
 - Publishing Objects Statically 31-6
 - Publishing Objects Dynamically 31-7
 - Handling Remote Objects 31-7
 - Registering Remote Objects Statically 31-7
 - Registering Remote Objects Dynamically 31-8
 - Publishing Remote Objects 31-8
 - Publishing Objects Securely 31-8
 - Unpublishing Objects 31-9
- Accessing Published Objects 31-10
 - Using CTMClient 31-10
 - Using CTMClientProxy 31-11
 - Using CTMCall 31-13
 - Changing CTM Client Properties 31-14
 - Using CTMConstants 31-15
 - Using the CTM Configuration File 31-15
 - Handling CTM Exceptions 31-17
- Handling Special Requirements 31-19
 - Implementing Secure CSTM Clients 31-19
 - Running Registry Server as a Separate Process 31-21
 - Registering the CSTM Port 31-21
 - Using SOAP Encoding With CSTM 31-21
 - Using the IMarshal Interface 31-22
 - Using marshalMethodAndArgs 31-23
 - Using unmarshalMethodAndArgs 31-23
 - Using unmarshalReturnValue 31-24

CISCO CONFIDENTIAL

Using marshalReturnValue	31-24
Using IMarshal's Register Method	31-24
Performing CSTM File Transfers	31-25
CTMFileTransfer Client Side Functionality	31-26
CTMFileTransfer Server Side Functionality	31-27
About CTMFileTransferException	31-27
Retrieving HTTP Errors	31-28
Using the CTMTest Tools and Samples	31-28
Creating a Custom Test File	31-29
Publishing a Test Object	31-29
Unpublishing a Test Object	31-29
Accessing a Test Method Using CTMClient	31-30
Accessing a Test Method Using CTMClientProxy	31-30
Accessing a Test Method Using CTMCall	31-31
Testing CSTM Communications	31-31
Using the Sample TestClass	31-32
Using the CSTM Samples	31-32
Testing Parameter Passing	31-33
Testing for Timeout Errors	31-33
Testing Multiple Clients	31-34
Guidelines for Using CSTM	31-34

CHAPTER 32

Using Package Support Updater	32-1
Understanding PSU	32-2
Using PSU with Your Application	32-2
Integrating Applications	32-2
Adding New Tags in INFO Files	32-3
Registering with PSU	32-3
Implementing Package Adapter and Package Descriptor Interfaces	32-4
Using the PSU Command Line Tools	32-4
Backing Up the Server	32-5
Releasing Package Updates	32-6
Uninstalling Device Support Packages	32-6
Working with Software Center	32-6
Performing Software Updates	32-7
Performing Device Updates	32-8
Scheduling Device Downloads	32-9
Viewing Activity Logs	32-9
Scheduled Job Details	32-10

CISCO CONFIDENTIAL

Event Logs 32-10

CHAPTER 33

Using Common Incremental Device Support 33-1

Understanding CIDS 33-3

SDI Component 33-3

Abstraction Groups 33-3

Runtime Architecture 33-4

CHAPTER 34

Using the Licensing APIs 34-1

Understanding CWCS Licensing 34-1

Using Licensing UI 34-2

Understanding CWCS Licensing APIs 34-4

CWCS Licensing Classes 34-4

LicensedFeature 34-4

LicenseManager 34-5

LicensePAK 34-5

Error Codes Generated by APIs 34-5

Integrating CWCS Licensing APIs 34-6

JavaDoc 34-6

License Installation 34-6

PAK and PIN 34-7

Handling Multiple Licenses 34-7

Interpreting PIN 34-8

PIN Format 34-9

Understanding License Framework 34-10

Flowcharts 34-10

Using Licensing Framework With Applications 34-12

Install 34-12

CW Home Page 34-12

Runtime Calls 34-12

License SDK 34-13

Data Architecture 34-13

License File Format 34-13

License File Format for Common Services 3.0 34-15

Alternate License File Format 34-16

Proof-of-Purchase (POP) 34-17

License CLI 34-19

CISCO CONFIDENTIAL

GLOSSARY

INDEX

CISCO CONFIDENTIAL



Preface

This manual describes how to integrate your application with CiscoWorks Common Services (CWCS). It provides an overview of CWCS, instruction in how to use individual CWCS components, API and CLI summaries, and references to outside sources for more detailed information.

Audience

This guide is written for Cisco software engineers who want to integrate their web-based network management applications or tools with CiscoWorks Common Services. These engineers should have a working knowledge of Java, PERL, or another web-based programming language, and HTML. They should also have some experience developing servlets and applets.

Conventions

This document uses the following conventions:

Item	Convention
Commands and keywords	boldface font
Variables for which you supply values	<i>italic font</i>
Displayed session and system information	screen font
Information you enter	boldface screen font
Variables you enter	<i>italic screen font</i>
Menu items and button names	boldface font
Selecting a menu item	Option>Network Preferences

This guide also uses the following conventions:

- If items such as buttons or menu options are grayed out on screens, it means that you do not have permission to use these items.
- Path names use the UNIX conventions. For Windows platforms, the path separator is '\ ' instead of '/' and the elements of the class path are separated with ';' instead of '.'.

CISCO CONFIDENTIAL

Command syntax notation conventions include the following:

Notation	Description
Braces ({})	Indicate a required choice.
Brackets, square ([])	Indicate an optional element.
Vertical bars ()	Indicate separate, mutually exclusive elements.

**Note**

Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the publication.

**Caution**

Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation

**Note**

Although every effort has been made to validate the accuracy of the information in the printed and electronic documentation, you should also review the CWCS documentation on Cisco.com for any updates.

The guides shown in [Table 1](#) are available from the CWCS 3.0 SDK Portal.

Table 1 **Documentation Available in the CWCS SDK**

Book Name	Description
SDK Documentation	
SDK Developer's Guide for CiscoWorks Common Services 3.0.5.	Explanation of Common Services and how to integrate with it. This document.
End-User Documentation	
<i>User Guide for CiscoWorks Common Services 3.0.5</i>	Describes basic tasks provided by CWCS.
<i>Installation and Setup Guide for Common Services 3.0 (Includes CiscoView) on Windows</i>	Describes the CWCS installation procedure for Windows.
<i>Installation and Setup Guide for CiscoWorks Common Services 3.0 (Includes CiscoView) on Solaris</i>	Describes the CWCS installation procedure for Solaris

CISCO CONFIDENTIAL

Document Organization

The *SDK Developer's Guide for CiscoWorks Common Services 3.0.5* is organized as follows:

Part 1, "About CWCS"

- [Chapter 1, "Introducing CWCS"](#) provides an overview of CWCS and its SDK.
- [Chapter 2, "FAQs and Programming Hints"](#) provides answers to frequently asked questions (FAQs) about and tips for programmers using CWCS.
- [Chapter 3, "Understanding the CWCS Directory Structure"](#) provides a detailed description of the CWCS directory structure.
- [Chapter 4, "Understanding the CWCS Execution Environment"](#) describes the CWCS execution environment and provides guidelines for creating applications to run in this environment.
- [Chapter 5, "Getting Started with CWCS"](#) provide basic information on how to integrate your applications with CWCS.

Part 2, "Using Shared Services"

- [Chapter 6, "Using Shared Services"](#) explains the concept of CWCS Shared Services and provides basic information about the Shared Services components.
- [Chapter 7, "Using the CiscoWorks Home Page"](#) describes how to integrate your application with the CiscoWorks Home Page.
- [Chapter 8, "Using Web Servers and Servlet Engines"](#) describe how to use the CWCS Web Server and servlet engine with your CWCS-based applications.
- [Chapter 9, "Integrating Applications with CMIC"](#) explains how to use CMIC to integrate your application services and functions with services provided by other applications and components on the network.
- [Chapter 10, "Using the Security System"](#) explains how to integrate with CWCS authorization and security features.
- [Chapter 11, "Using the Database APIs"](#) explains how to create a custom database for your application and use the CWCS database APIs to install and configure it.
- [Chapter 12, "Using Backup and Restore"](#) explains how to use the CWCS database backup and restore components.
- [Chapter 13, "Using the Core Client Registry"](#) describes how to use CCR to manage installation, upgrade, patching and uninstall of the MDC and Core modules.
- [Chapter 14, "Using the Device Credentials Repository"](#) explains how to implement and share common repositories of device information.
- [Chapter 15, "Using the Core Logging API"](#) provides guidelines on how to save and share MDC/Core log and audit messages in shared files from both Java and C++.
- [Chapter 16, "Adding Online Help"](#) describes the CWCS help engine and how to add help for your application to the help system suite.
- [Chapter 17, "Using the Daemon Manager"](#) explains how to use the Daemon Manager start, monitor and restart processes, including long-running, dependent and transient processes.
- [Chapter 18, "Using the Job and Resource Manager"](#) shows how to use JRM to schedule jobs and track locked resources by name.
- [Chapter 19, "Using Event Services Software"](#) explains how to use the ESS asynchronous messaging service to enable distributed, loosely coupled interprocess communications.

CISCO CONFIDENTIAL

- [Chapter 20, “Using the Event Distribution System”](#) explains the deprecated EDS messaging service.
- [Chapter 21, “Using the Installation Framework”](#) provides complete information on using CWCS’s comprehensive set of application installation, uninstallation, and patching tools.
- [Chapter 22, “Using the Java Plug-in”](#) explains how to speed up execution by using Sun’s Java Plug-in with your application.
- [Chapter 23, “Using the Diagnostic and Support Utilities”](#) describes how to use the CWCS Collect Server Info, MDC Support, SNMP Set and Walk, Packet Capture and Logrot utilities.
- [Chapter 24, “Using SNMP Services”](#) describes the SNMPv3 and SNMPv1/v2c support libraries provided with CWCS.
- [Chapter 25, “Using NT Services”](#) describes how CWCS supports on Windows the communication services commonly available on UNIX. These services include TFTP, RCP, TELNET and Syslog.
- [Chapter 26, “Using Device Center”](#) describes how to use the Device Center interface to organize all information, tasks and reports for the device at a single location.
- [Chapter 27, “Using Product Instance Device Mapping”](#) describes how to use the Product Instance Device Mapping APIs and Product Instance Device Mapping North Bound APIs.
- [Chapter 28, “Integrating Applications With Device Selector”](#) describes how to integrate the applications with the new and enhanced Device Selector.

Part 3, “Using Per-Product Services”

- [Chapter 29, “Using Per-Product Services”](#) explains the concept of CWCS Per-Product Services and provides basic information about the Per-Product components.
- [Chapter 30, “Using Object Grouping Services”](#) explains how to use OGS to create, manage and share groups of objects.
- [Chapter 31, “Using the Common Services Transport Mechanism”](#) provides detailed information on using CSTM to handle all kinds of programmatic communications, including inter-process, intra-process, and remote procedure calls.
- [Chapter 32, “Using Package Support Updater”](#) explains how to use PSU to download and install device packages and software updates for your application.
- [Chapter 33, “Using Common Incremental Device Support”](#) explains how to use the CIDS and PSU mechanisms to update your application’s device support without re-installation.
- [Chapter 34, “Using the Licensing APIs”](#) shows how to use the CWCS licensing framework to implement a licensing model for your application, and to install, update, and retrieve license information.

CISCO CONFIDENTIAL

Obtaining Documentation

Cisco documentation and additional literature are available on Cisco.com. Cisco also provides several ways to obtain technical assistance and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

You can access the most current Cisco documentation at this URL:

<http://www.cisco.com/techsupport>

You can access the Cisco website at this URL:

<http://www.cisco.com>

You can access international Cisco websites at this URL:

http://www.cisco.com/public/countries_languages.shtml

Product Documentation DVD

The Product Documentation DVD is a library of technical product documentation on a portable medium. The DVD enables you to access installation, configuration, and command guides for Cisco hardware and software products. With the DVD, you have access to the HTML documentation and some of the PDF files found on the Cisco website at this URL:

<http://www.cisco.com/univercd/home/home.htm>

The Product Documentation DVD is created monthly and is released in the middle of the month. DVDs are available singly or by subscription. Registered Cisco.com users can order a Product Documentation DVD (product number DOC-DOCDVD= or DOC-DOCDVD=SUB) from Cisco Marketplace at the Product Documentation Store at this URL:

<http://www.cisco.com/go/marketplace/docstore>

Ordering Documentation

You must be a registered Cisco.com user to access Cisco Marketplace. Registered users may order Cisco documentation at the Product Documentation Store at this URL:

<http://www.cisco.com/go/marketplace/docstore>

If you do not have a user ID or password, you can register at this URL:

<http://tools.cisco.com/RPF/register/register.do>

Documentation Feedback

You can provide feedback about Cisco technical documentation on the Cisco Technical Support & Documentation site area by entering your comments in the feedback form available in every online document.

CISCO CONFIDENTIAL

Cisco Product Security Overview

Cisco provides a free online Security Vulnerability Policy portal at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

From this site, you will find information about how to do the following:

- Report security vulnerabilities in Cisco products
- Obtain assistance with security incidents that involve Cisco products
- Register to receive security information from Cisco

A current list of security advisories, security notices, and security responses for Cisco products is available at this URL:

<http://www.cisco.com/go/psirt>

To see security advisories, security notices, and security responses as they are updated in real time, you can subscribe to the Product Security Incident Response Team Really Simple Syndication (PSIRT RSS) feed. Information about how to subscribe to the PSIRT RSS feed is found at this URL:

http://www.cisco.com/en/US/products/products_psirt_rss_feed.html

Reporting Security Problems in Cisco Products

Cisco is committed to delivering secure products. We test our products internally before we release them, and we strive to correct all vulnerabilities quickly. If you think that you have identified a vulnerability in a Cisco product, contact PSIRT:

- For emergencies only—security-alert@cisco.com

An emergency is either a condition in which a system is under active attack or a condition for which a severe and urgent security vulnerability should be reported. All other conditions are considered nonemergencies.

- For nonemergencies—psirt@cisco.com

In an emergency, you can also reach PSIRT by telephone:

- 1 877 228-7302
- 1 408 525-6532

**Tip**

We encourage you to use Pretty Good Privacy (PGP) or a compatible product (for example, GnuPG) to encrypt any sensitive information that you send to Cisco. PSIRT can work with information that has been encrypted with PGP versions 2.x through 9.x.

Never use a revoked encryption key or an expired encryption key. The correct public key to use in your correspondence with PSIRT is the one linked in the Contact Summary section of the Security Vulnerability Policy page at this URL:

http://www.cisco.com/en/US/products/products_security_vulnerability_policy.html

The link on this page has the current PGP key ID in use.

If you do not have or use PGP, contact PSIRT to find other means of encrypting the data before sending any sensitive material.

CISCO CONFIDENTIAL

Product Alerts and Field Notices

Modifications to or updates about Cisco products are announced in Cisco Product Alerts and Cisco Field Notices. You can receive Cisco Product Alerts and Cisco Field Notices by using the Product Alert Tool on Cisco.com. This tool enables you to create a profile and choose those products for which you want to receive information.

To access the Product Alert Tool, you must be a registered Cisco.com user. (To register as a Cisco.com user, go to this URL: <http://tools.cisco.com/RPF/register/register.do>) Registered users can access the tool at this URL: <http://tools.cisco.com/Support/PAT/do/ViewMyProfiles.do?local=en>

Obtaining Technical Assistance

Cisco Technical Support provides 24-hour-a-day award-winning technical assistance. The Cisco Technical Support & Documentation website on Cisco.com features extensive online support resources. In addition, if you have a valid Cisco service contract, Cisco Technical Assistance Center (TAC) engineers provide telephone support. If you do not have a valid Cisco service contract, contact your reseller.

Cisco Technical Support & Documentation Website

The Cisco Technical Support & Documentation website provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The website is available 24 hours a day at this URL:

<http://www.cisco.com/techsupport>

Access to all tools on the Cisco Technical Support & Documentation website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a user ID or password, you can register at this URL:

<http://tools.cisco.com/RPF/register/register.do>



Note

Use the **Cisco Product Identification Tool** to locate your product serial number before submitting a request for service online or by phone. You can access this tool from the Cisco Technical Support & Documentation website by clicking the **Tools & Resources** link, clicking the **All Tools (A-Z)** tab, and then choosing **Cisco Product Identification Tool** from the alphabetical list. This tool offers three search options: by product ID or model name; by tree view; or, for certain products, by copying and pasting **show** command output. Search results show an illustration of your product with the serial number label location highlighted. Locate the serial number label on your product and record the information before placing a service call.



Tip

Displaying and Searching on Cisco.com

If you suspect that the browser is not refreshing a web page, force the browser to update the web page by holding down the Ctrl key while pressing F5.

To find technical information, narrow your search to look in technical documentation, not the entire Cisco.com website. On the Cisco.com home page, click the **Advanced Search** link under the Search box

CISCO CONFIDENTIAL

and then click the **Technical Support & Documentation** radio button.

To provide feedback about the Cisco.com website or a particular technical document, click **Contacts & Feedback** at the top of any Cisco.com web page.

Submitting a Service Request

Using the online TAC Service Request Tool is the fastest way to open S3 and S4 service requests. (S3 and S4 service requests are those in which your network is minimally impaired or for which you require product information.) After you describe your situation, the TAC Service Request Tool provides recommended solutions. If your issue is not resolved using the recommended resources, your service request is assigned to a Cisco engineer. The TAC Service Request Tool is located at this URL:

<http://www.cisco.com/techsupport/servicerequest>

For S1 or S2 service requests, or if you do not have Internet access, contact the Cisco TAC by telephone. (S1 or S2 service requests are those in which your production network is down or severely degraded.) Cisco engineers are assigned immediately to S1 and S2 service requests to help keep your business operations running smoothly.

To open a service request by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411

Australia: 1 800 805 227

EMEA: +32 2 704 55 55

USA: 1 800 553 2447

For a complete list of Cisco TAC contacts, go to this URL:

<http://www.cisco.com/techsupport/contacts>

Definitions of Service Request Severity

To ensure that all service requests are reported in a standard format, Cisco has established severity definitions.

Severity 1 (S1)—An existing network is “down” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Severity 2 (S2)—Operation of an existing network is severely degraded, or significant aspects of your business operations are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Severity 3 (S3)—Operational performance of the network is impaired while most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Severity 4 (S4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

CISCO CONFIDENTIAL

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- The *Cisco Product Quick Reference Guide* is a handy, compact reference tool that includes brief product overviews, key features, sample part numbers, and abbreviated technical specifications for many Cisco products that are sold through channel partners. It is updated twice a year and includes the latest Cisco channel product offerings. To order and find out more about the *Cisco Product Quick Reference Guide*, go to this URL:

<http://www.cisco.com/go/guide>

- Cisco Marketplace provides a variety of Cisco books, reference guides, documentation, and logo merchandise. Visit Cisco Marketplace, the company store, at this URL:

<http://www.cisco.com/go/marketplace/>

- Cisco Press publishes a wide range of general networking, training, and certification titles. Both new and experienced users will benefit from these publications. For current Cisco Press titles and other information, go to Cisco Press at this URL:

<http://www.ciscopress.com>

- *Packet* magazine is the magazine for Cisco networking professionals. Each quarter, *Packet* delivers coverage of the latest industry trends, technology breakthroughs, and Cisco products and solutions, as well as network deployment and troubleshooting tips, configuration examples, customer case studies, certification and training information, and links to scores of in-depth online resources. You can subscribe to *Packet* magazine at this URL:

<http://www.cisco.com/packet>

- *Internet Protocol Journal* is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the *Internet Protocol Journal* at this URL:

<http://www.cisco.com/ipj>

- Networking products offered by Cisco Systems, as well as customer support services, can be obtained at this URL:

<http://www.cisco.com/en/US/products/index.html>

- Networking Professionals Connection is an interactive website where networking professionals share questions, suggestions, and information about networking products and technologies with Cisco experts and other networking professionals. Join a discussion at this URL:

<http://www.cisco.com/discuss/networking>

- “What’s New in Cisco Documentation” is an online publication that provides information about the latest documentation releases for Cisco products. Updated monthly, this online publication is organized by product category to direct you quickly to the documentation for your products. You can view the latest release of “What’s New in Cisco Documentation” at this URL:

<http://www.cisco.com/univercd/cc/td/doc/abtnicd/136957.htm>

- World-class networking training is available from Cisco. You can view current offerings at this URL:

<http://www.cisco.com/en/US/learning/index.html>

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL



PART 1

About CWCS



CISCO CONFIDENTIAL

CHAPTER 1

Introducing CWCS

CiscoWorks Common Services (Common Services) represents a common set of management services including a collection of subsystems, execution environments, engines, and shared code libraries, and serves as a software platform to web-based network management applications.

All CiscoWorks products use and depend on Common Services. Common Services provides a foundation for CiscoWorks applications to share a common model for data storage, login, user role definitions, access privileges, security protocols, as well as navigation. It creates a standard user experience for all management functions. It also provides a common framework for all basic system-level operations such as installation, data management including backup-restore and import-export, event and message handling, and job and process management.

The CWCS SDK is for developers who want to create CWCS-based applications or update existing applications to take advantage of new CWCS features.

The following topics introduce the main concepts of CWCS:

- [CWCS Release Model](#)
- [Benefits of Using CWCS](#)
- [What's New in CWCS](#)
- [What's New in This Guide](#)
- [Understanding the CWCS Structure](#)
- [Using CWCS Components](#)
- [How CWCS is Distributed](#)
- [Third-Party Tools](#)
- [Where to Find the CWCS SDK](#)
- [For Further Assistance](#)

CWCS Release Model

Every CWCS release is provided in these forms:

1. **CWCS-R Release:** This is essentially a replacement for the Common Management Foundation (CMF)-based CD One. This release is a physical CD containing the CWCS-R installable image, plus the latest versions of CiscoView and the Integration Utility (IU, formerly known as NMIM) as optional installs. Developers can modify the installation wrapper to provide options for installing Common Services alone, or Common Services plus either CiscoView or the Integration Utility (or both). This is the standard installable image used by most application teams within NMTG.

CISCO CONFIDENTIAL

2. **CWCS-SRC:** The CiscoWorks Common Services source toolkit. This is a downloadable file containing source code and binaries for integration with applications' installable packages. It contains CWCS components designated as "open source", which includes the User Interface Infrastructure (UII), Object Grouping Services (OGS), and the Common Services Transport Mechanism (CSTM). This release is provided for the use of NMTG application teams creating customized "CD-One-like" Common Services CDs.

The CWCS- R release is developed, tested, packaged and managed under the direct control of the CWCS team. Key stakeholders set the requirements for each release, and releases take place on a defined schedule agreed upon among the CWCS team and stakeholders. All fixes, patches, updates and other coding are the responsibility of the CWCS team.

The CWCS-SRC release is developed by individual application teams as well as by dedicated groups within the CWCS team. After delivery, SRC components may be modified by application teams to suit their needs, and must be installed with individual applications on a per-product basis.

While some CWCS components delivered in CWCS-R are also designated as SRC, the instances of SRC components delivered with and installed as part of CWCS-R are for the exclusive use of Common Services. Application teams may not overwrite the shared, common version of the component with a version of their own. For example: The Object Grouping Service (OGS) is an SRC component delivered as part of CWCS-R. Application teams who want to use OGS must create one and install it with their application.

CWCS-R releases follow a release train strategy to allow for phased delivery:

- Major releases mark the beginning of a new release train. These releases contain major architectural changes, are content-driven, and ship about once a year.
- Minor releases continue a release train and may be shipped at other times.
- Patches are provided for critical problems.
- Numbered Service Packs are released when sufficient patches accumulate.

Each CWCS-R release train follows these assumptions:

- One Concept Commit and one Execution Commit for the release.
- One PRD, one program plan, and one system functional specification.
- Detailed functional specifications for each major and minor feature.
- External testing period prior to FCS with applications that use new features.

CWCS-SRC components are provided in at least one working version per CWCS-R release, at the time of the CWCS-R release. Alternative or improved versions of SRC components may or may not be available at other times, and may undergo design changes, fixes, re-coding and other updates without reference to the CWCS-R schedule. Any modified versions of SRC components, and fixes or updates to them, are the sole responsibility of the application teams modifying them; they are, in fact, deliverables of the applications requiring them.

All third-party components are licensed to ship with CWCS-R and CWCS-SRC. Any distribution of these components outside of CWCS may not be covered by the license. Any questions about licensing should be sent to the cmf23-dev alias.

CISCO CONFIDENTIAL

Benefits of Using CWCS

Application development teams derive the following benefits from using CWCS:

- Use of a common, unified infrastructure for next-generation network management products and solutions allows for better integration across such products. For example, users see a single security model, desktop GUI, and application launch across applications.
- Enhanced flexibility for meeting special requirements by re-using SRC components in an “open source” manner.
- Existing, proven code has already been debugged and optimized, resulting in improved product stability and performance.
- Existing code can be reused, resulting in faster time-to-market for new applications.
- Unnecessary duplication of effort is avoided, resulting in more efficient use of engineering resources.
- Significant leverage across Cisco and third-party development teams is possible.

What's New in CWCS

This CWCS release provides the following new features and enhancements:

- **CiscoWorks Home Page (CWHP):** Provides a central, customizable launch point for all installed CWCS-based applications as well as non-Cisco application links with HTML-based login.
- **Common Services Home (CS Home):** Serves as the dashboard for the Common Services application, and provides launch points for the frequently used functions in Common Services application, status summary of jobs, status of security configurations, backup schedule, Online User information and information on DCA mode and number of devices.
- **Device and Credentials Repository (DCR):** Provides shared device information and credentials for CWCS-based applications running on multiple servers. It also supports standalone repositories.
- **CWCS Licensing Framework:** Allows FlexLM-based licensing on a wide variety of models.
- **Cisco Management Integration Center (CMIC):** Supplies a consolidated service registry that allows applications to find and integrate with other applications and services residing outside the server machine. It also allows customers to set up and launch third-party applications from the CiscoWorks Home Page.
- **Online Help:** Support for the UE-compatible version 2.0 of the Cisco online help engine.
- **LMS Setup Center:** Allows you to configure the Security settings, System settings, Data Collection settings, Data Collection Schedule settings, and Data Purge settings of all CiscoWorks applications in a centralized place.
- **Software Center:** Using the Common Incremental Device Support (CIDS) and Package Support Utility (PSU) components, Software Center allows applications to automatically locate, download and install device-support packages and software updates.
- **Device Center:** Provides an interface to invoke any application tool on a selected device from a single location. It also provides summary information needed to troubleshoot a selected device.
- **Installation Framework:** Improved and streamlined, the Installation Framework is compatible with the new Licensing Framework, and now supports workflow customization on Solaris. Support for the little-used Express install and remote upgrade options has been withdrawn, and “Typical” installs now support simpler options.

CISCO CONFIDENTIAL

- **Security System:** This important subsystem provides a single application interface for AAA security, secure and encrypted communications between clients and servers and between servers and devices, and libraries that enable encryption of data on a server.
- **Object Grouping Services:** Now supports shared and common groups.
- **Backup and Restore Framework:** Includes both Core and CMF-based backup and restore. CMF-based backup-manifest file locations have been changed to support CWCS database enhancements, and permit restore from previous versions of CMF, eliminating the need for Remote Upgrade capabilities.
- **Job and Resource Manager (JRM):** Now provides separate tracking for instances of a recurring job, updated job control APIs, and publishes its events to ESS and EDS.
- **Diagnostic and Support Utilities:** The JET, JT and Logrot tools have been added to the diagnostic and support tools suite.
- **Third Party Components:** Upgrades to most third-party components, including the Java Runtime Environment and JPI, the Sybase database, TibcoRV, the XML parsers and associated tools, and others. For a complete list of these tools and the versions supported in this release, see the [“Third-Party Tools” section on page 1-10](#).
- **Cumulative bug fixes and patches provided for all previous CWCS releases.**
- **Support for:**
 - Internet Explorer 6
 - Netscape 7.x and Mozilla 1.7. 13
 - Solaris 8 and 9.
 - Sybase 9.0.0.
 - Windows 2000 Professional
 - Windows 2000 Server
 - WIndows 2000 Advanced Server
 - Windows Server 2003
 - Windows 2003 R2 Server
 - SNMPv3 authNoPriv.
 - IPv6
- **As of this release, CWCS no longer supports:**
 - Netscape 4.7x.
 - Solaris 7.

**Caution**

Be sure to review the appropriate topics in this document to ensure that your code works with CWCS 3.0. For an inventory of all new and changed components and their related documentation, see the [“Using CWCS Components” section on page 1-7](#).

CISCO CONFIDENTIAL

What's New in This Guide

This SDK Developer's Guide contains the following new chapter:

- [Chapter 28, "Integrating Applications With Device Selector"](#)

Updates have also been made to topics in other chapters, as well as to code samples included in the SDK downloads.

Please refer to the CWCS SDK portal at https://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=2537 for links to SDK downloads, documentation, and code samples.

This document does *not* include information that is available in:

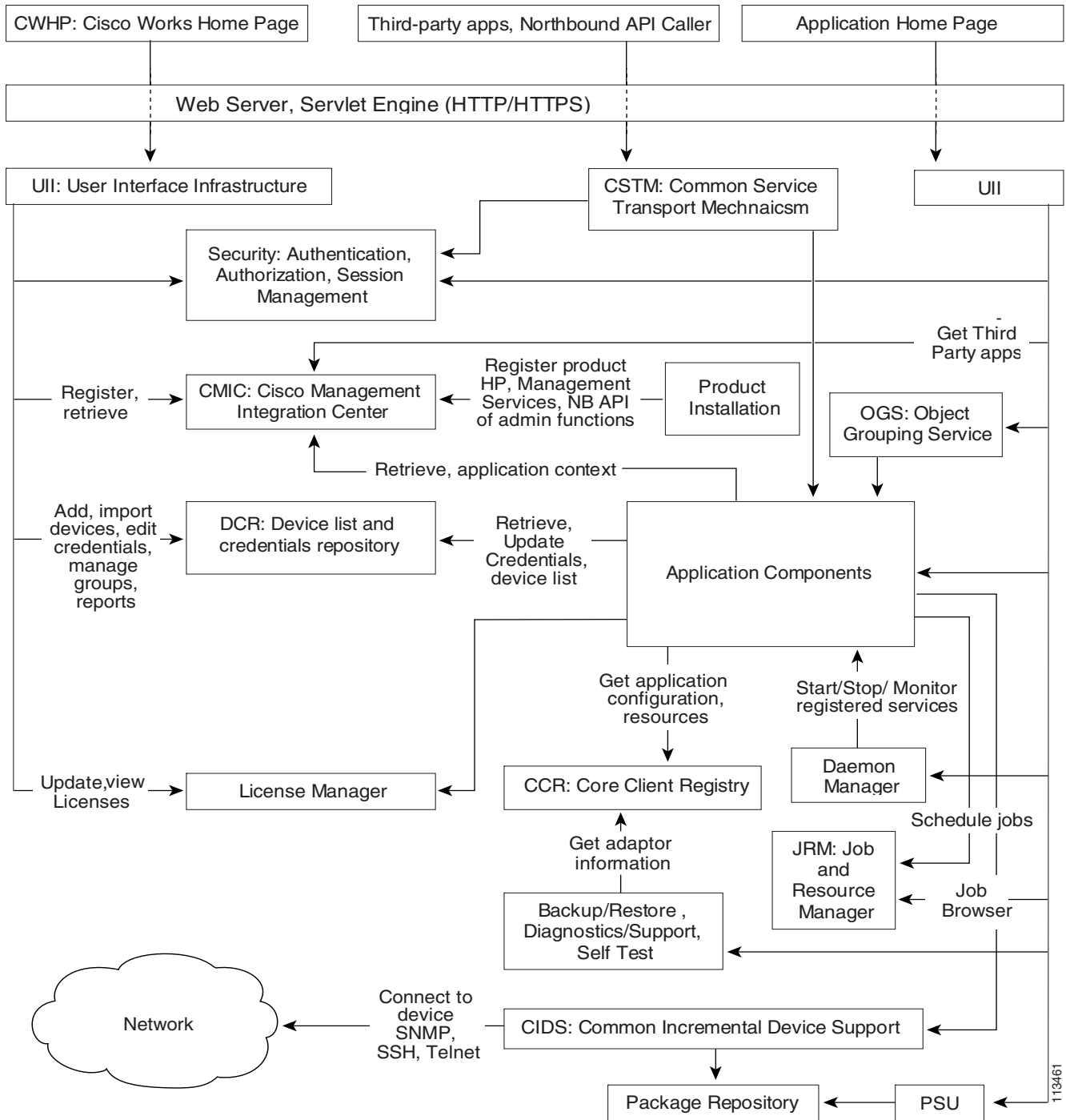
- The CiscoWorks Common Services 3.0.5 user documentation. To access this documentation set, see:
 - The online help for any installed CWCS-based application. To view the online help, launch the application and select **Help**.
 - The HTML version of the user documentation, published internally at the following URL: http://www.cisco.com/en/US/products/sw/cscowork/ps3996/products_user_guide_book09186a00806feda7.html.
- The User Interface Infrastructure SDK. To access this SDK, see the UE/UII web site at the URL <http://picasso>, or download a copy from EDCS at http://wwwin-eng.cisco.com/Eng/ENM/UE_UII/.

Understanding the CWCS Structure

CWCS is designed as a multi-component framework, as shown in [Figure 1-1](#).

CISCO CONFIDENTIAL

Figure 1-1 CWCS Structural Overview



CISCO CONFIDENTIAL

Using CWCS Components

CWCS provides the components shown in [Table 1-1](#). Components that are new in this release of CWCS appear at the beginning of the table and are highlighted in **bold**. Items present in CWCS, but not documented in this Guide, are highlighted in *italics*.

The CWCS directory structure is described in [Chapter 3, “Understanding the CWCS Directory Structure”](#). Basic development information for all components is provided in [Chapter 6, “Using Shared Services”](#) and in [Chapter 29, “Using Per-Product Services”](#). Details on individual components are documented in specific chapters listed within the Description in [Table 1-1](#).

Table 1-1 CiscoWorks Common Services Components

Component	Description
Cisco Management Integration Center (CMIC)	CMIC is a repository that allows CWCS-based and third-party applications to register the services they provide and look up the services provided by other applications. It allows better integration among applications and services residing anywhere in the network. See Chapter 9, “Integrating Applications with CMIC” .
CiscoWorks Home Page (CWHP)	CWHP provides your customers with a single, user-customizable web page they can use log in and launch your application, consolidate launch points for other applications, and add links to sites they use frequently. See Chapter 7, “Using the CiscoWorks Home Page” .
Common Incremental Device Support (CIDS)	CIDS provides applications with “drop in” support for new device types, eliminating the need for your customers to re-install the entire application. See Chapter 33, “Using Common Incremental Device Support” .
Device and Credentials Repository (DCR)	DCR provides a secure, sharable repository of critical device ID and credentials information. See Chapter 14, “Using the Device Credentials Repository” .
Device Center	Device Center gives your customers a “device-centric” view of their installed application suite by letting them run registered tasks from any application against a selected device. Instead of needing to know in advance which application performs which task, customers can select tasks arranged around the device in which they are interested. See Chapter 26, “Using Device Center” .
Licensing Framework	The licensing framework lets your application install, update and retrieve information about customer licenses. The framework includes APIs and FLEXlm utilities that you can use to implement a wide variety of licensing models. See Chapter 34, “Using the Licensing APIs” .
Package Support Updater (PSU)	PSU lets your application check for software and device support updates, download them to the CWCS server, and install them. See Chapter 32, “Using Package Support Updater” .
Backup and Restore Framework	This framework provides separate database backup and restore frameworks for Management Center (MC) applications and for all other applications. See Chapter 12, “Using Backup and Restore” .
Common Services Transport Mechanism (CSTM)	CSTM (formerly CTM) provides a consistent, simple, and platform-agnostic method for handling all types of application-to-application communications. It follows non-proprietary standards, and does not impose protocol, object model, or encoding restrictions on either communicating application. See Chapter 31, “Using the Common Services Transport Mechanism” .
CORBA Infrastructure	The CORBA infrastructure allows for object-oriented client/server communication.
Daemon Manager	Provides reliable, ordered execution services for server processes, desktop administrative interfaces, and command-line interfaces (for debugging). See Chapter 17, “Using the Daemon Manager” .

CISCO CONFIDENTIAL**Table 1-1 CiscoWorks Common Services Components (continued)**

Component	Description
Database APIs	Sybase Adaptive Server Anywhere (ASA) is bundled with CWCS. ASA functions as an embedded relational database, and requires very little administration. A single CWCS installation has the capability to support multiple databases for multiple applications. Database access is provided by JDBC in Java, ODBC in C and C++, and DBI in Perl. See Chapter 11, “Using the Database APIs” .
Diagnostic and Support Utilities	This utility suite helps you and TAC staff diagnose and solve customer problems quickly. Included are the Collect Server Info, MDC Support, SNMP Set and Walk, Packet Capture, and logrot. See Chapter 23, “Using the Diagnostic and Support Utilities” .
Event Services Software (ESS)	Handles all event messaging using a separate server bus (TibCo Rendezvous), a separate client bus (LWMS), and a gateway that transfers messages between the two automatically. See Chapter 19, “Using Event Services Software” . Note <i>The EDS event messaging system is deprecated.</i> For reference purposes only, EDS documentation has been retained in Chapter 17, “Using the Event Distribution System” .
Hidden Tools (dbreader and log file viewer)	<ul style="list-style-type: none"> dbreader: <code>http://server-name/dbreader/dbreader.html?eudora="cautourl"http://server-name/dbreader/dbreader.html</code> log file browser: <code>http://server_name:port/cgi-bin/searchLog.pl?eudora="autourl"http://server_name:port/cgi-bin/searchLog.pl</code>
Installation Framework	Allows you to install, uninstall, and patch your application. See Chapter 21, “Using the Installation Framework” .
Java Plug-in (JPI)	Allows you to deploy Java 2-based applets on your CiscoWorks web pages on Windows- and Solaris- based browsers. See Chapter 22, “Using the Java Plug-in”
Java Runtime Environment (JRE)	CWCS provides both Java 1 and 2 Runtime Environments.
Java SNMP Engine	The Java interface for the Simple Network Management Protocol (SNMP) is supported.
Job and Resource Manager	Provides services for scheduling jobs to run in the background and also locking functionality for network devices. See Chapter 18, “Using the Job and Resource Manager” .
NT Services	These services provide support for Windows versions of the communication functions commonly available on Solaris, including TFTP, RCP and Syslog. See Chapter 25, “Using NT Services”
Object Grouping Services (OGS)	OGS provides a generic means for creating, managing and sharing groups of objects of any type. See Chapter 30, “Using Object Grouping Services” .
Online Help System	Provides a customized help engine infrastructure, allowing applications to use a common mechanism to provide online help to customers. See Chapter 16, “Adding Online Help” .
Perl Interpreter	An interpreter for the Perl scripting language used in Web applications.
Security System	This system allows you to protect data, applications and server-to-device communications from inadvertent or malicious access. User-level security includes allocation of roles based on defined task-execution privileges. See Chapter 10, “Using the Security System” .
SNMP Services	These services provide access to SNMPv3’s enhanced security features, as well as supporting the existing SNMPv1 and SNMPv2c functions. See Chapter 24, “Using SNMP Services” .

CISCO CONFIDENTIAL**Table 1-1 CiscoWorks Common Services Components (continued)**

Component	Description
User Interface Infrastructure (UII)	UII provides the servlets and APIs needed to implement the CiscoWorks Home Page and all other CWCS user interface elements. See the <i>SDK Developer's Guide to UI Infrastructure</i> , accessible from the UE/UII web site at http://picasso , or http://www.win-eng.cisco.com/Eng/ENM/UE_UII/ .
Web Server and Servlet Engine	The CWCS web server and servlet engine use the Apache Web server on both UNIX and Windows platforms to provide the infrastructure for client/server communication. The Web server services HTTP requests from the client, and is also used to invoke CGI scripts/programs, applets, and servlets. A common servlet engine, Tomcat, runs Java servlet programs. See Chapter 8, "Using Web Servers and Servlet Engines".

How CWCS is Distributed

You can distribute CWCS with your application using the "CD-One" CD bundled with your application. In this case, you are delivering the CWCS-R+ distribution described in the "CWCS Release Model" section on page 1-1. This is the standard method of delivering CWCS, used by nearly all NMTG application teams.

This is the only supported option. If you need to discuss this option, contact the cmf23-dev alias.

Installation Interface Options

The CD One installation interface under the Custom install option presents the following choices:

- CiscoWorks Common Services
- CiscoView
- Integration Utility
- Typical Installation (all components)

For additional information on which packages are being installed, refer to the "Installing CWCS" section on page 5-3.

CD Image Structure

The Windows and Solaris CD image structures are different. Generally, the CD contains one or more installable units, suites, and the installer. Each installable unit publishes its properties, including names, versions, dependencies, and so on. Some properties are provided by the build or appended while the CD image is created. The installer is a separate application with its own release train.

For details about the Windows and Solaris CD structures, see the following topics:

- [CD Image Structure for Windows](#)
- [CD Image Structure for Solaris](#)

CISCO CONFIDENTIAL**CD Image Structure for Windows**

The following components are part of the Windows CD:

- Windows CD autorun configuration and executable
- InstallShield runtime
- Installer—Compiled installation scripts with the tool set that actually implements the functionality specified in this document.
- Table of contents—ASCII file that describes the contents of the CD, enumerates components available on this CD, as well as defaults for installation. It also provides release-specific information (for example, release name).
- Each suite, installable unit and package contains:
 - Properties
 - Optional Install scripts (compiled versions of component hooks)
 - Package/installable unit specific tools required at installation time.
- Compressed runtime tree (data1.cab file) for all packages.

CD Image Structure for Solaris

The following subdirectories are part of the Solaris CD:

- info—package properties, installation descriptions, and package hooks
- packages—Solaris packages

Solaris uses SVR4.

Third-Party Tools

Table 1-2 lists the third-party tools available with CWCS.

**Caution**

Tools marked “obsolete and deprecated” are included for backward compatibility only. Do not use these tools for future development.

CISCO CONFIDENTIAL**Table 1-2 CWCS Third-Party Tools**

Name	Vendor	Package	Description
3Magic Coffee Table 2.1.3	3Magic http://www.3magic.com/products/coffeetable/docs/index.html	CSCOgrid	Small, speedy, cross-platform and multi-VM Java components for display of columnar data. Compatible with and optimized for JDK 1.1 and 1.2. Includes Grid classes for Java.
Alfred XML parser 1.2a	Open Text Corporation http://www.jpackage.org/rpm.php?id=540	CSCOxrts	A small, fast, DTD-aware Java-based XML parser, especially suitable for use in Java applets. You can add XML support to your applets and applications without doubling their size. Aelfred consists of only two core class files, with a total size of about 26K, and requires very little memory to run. There is also a complete SAX (Simple API for XML) driver available in this distribution for inter-operability.
Apache Web Server 1.3.31	Apache http://www.apache.org/httpd.html	apache	Apache is a popular Web Server on the Internet. It is based on open source that is publicly maintained by the Apache HTTP Server project. Apacheweek is a useful reference for information about Apache. CWCS provides this Web Server on Solaris and Windows.
Apache Axis 1.1	Apache http://ws.apache.org/axis/	CSCOweb	Apache Axis is a SOAP engine framework used to implement Java Web Services. It is Jakarta-based and works as a Tomcat plug-in.
Blatmail for NT 2.1	Open Source http://sourceforge.net/projects/blat	CSCOxrts	Blat is a Win32 command line utility that sends eMail using the SMTP or NNTP protocols.
DynAPI/ LGPL 2.5.6	Open Source http://sourceforge.net/projects/dynapi/	CSCOxrts	DynAPI is a cross-browser Javascript library used to create Dynamic HTML components on web pages.
FLEXLM 9.2	Macrovision http://www.macrovision.com/products/legacy_products/flexlm/index.shtml	CSCOcore	Software license management system and utilities used to manage licenses available on a machine
IPC lib from ACE 5.3	Open Source http://www.cs.wustl.edu/~schmidt/ACE.html	CSCOxrts	Adaptive Communication Environment (ACE) is an object-oriented C++ framework for developing multithreaded applications that include network and inter-process communications.
IPSecPol 1.22	Microsoft http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/ipsecpol-o.asp	ipsec	A CLI tool for creating large or complex Internet Protocol Security (IPSec) policies in a local or remote registry. Can be run in batch mode.

CISCO CONFIDENTIAL**Table 1-2 CWCS Third-Party Tools (continued)**

Name	Vendor	Package	Description
Java 2 Runtime Environment 13.1_06 and 1.4.2_04	Sun http://www.sun.com/solaris/jre/download.html	sunjre, CSCOjre14	The Java™ Runtime Environment (also known as the Java Runtime or JRE) consists of the Java virtual machine, the Java platform core classes, and supporting files. It is the runtime part of the Java Development Kit that has no compiler, no debugger, and no tools. The JRE is the smallest set of executables and files that constitute the standard Java platform.
Java Desktop Suite 0.16.3	Sun http://java.sun.com/	TBD	TBD
Java Extension 1.3	Sun http://java.sun.com/	TBD	TBD
Java Plug-In 1.4.2_10	Sun http://java.sun.com/products/plugin/	CSCOplug	Permits applets to be run within a desktop browser. Part of the Java 2 Runtime Environment.
JavaMail for Solaris 1.2	Sun http://java.sun.com/products/javamail/	CSCOjre14	Provides a platform- and protocol-independent framework for building mail and messaging applications.
JClass Chart 4.0.0.J	Quest Software http://www.quest.com/jclass_desktopviews/chart.asp	CSCOjchart	Part of Quest's JClass DesktopViews, JClass Chart provides Java tools for business and scientific charts, rich text format for customizing labels or mixing images and URLs with text.
jConnect for JDBC 5.5	Sybase http://www.sybase.com/products/middleware/jconnectforjdbc	CSCOdcb	The jConnect for JDBC product provides high performance native access to all Sybase products including Adaptive Server Anywhere. It integrates with most popular Java RAD tools.
JDOM 1.0.8b	JDOM Project http://www.jdom.org	jdom	JDOM is a Java-based "document object model" for XML files. JDOM serves the same purpose as DOM, but is easier to use. It provides a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code.
JFC Swing classes 1.1, 1.2c	Sun http://java.sun.com/j2se/1.4.2/docs/guide/swing/index.html	CSCOswng2	The Swing classes are a set of class libraries provided as part of the Java 2 Platform to support building graphic user interfaces (GUIs) and graphics functionality for client applications.
JGL 3.1	Open Source http://www.recursionsw.com/jgl.htm	CSCOjgl	JGL is a generic collection library for Java, consisting of eleven optimized Java collections and more than 50 general purpose data algorithms. JGL-equivalent functionality is available in Java 2 (aka JDK 1.2) today.
JMS 1.02b	Sun http://java.sun.com/products/jms/	sunjre	The Java Message Service (JMS) API is a messaging standard that allows application components based on the Java 2 Platform to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

CISCO CONFIDENTIAL**Table 1-2 CWCS Third-Party Tools (continued)**

Name	Vendor	Package	Description
JOnAS 3.3.5	ObjectWeb Consortium http://jonas.objectweb.org	CSCOxrts	The Java Open Application Server (JOnAS) is a pure Java open-source implementation of the EJB (Enterprise JavaBeans) specification, supporting Java Web Services.
JScrollPane 1.5.1	http://www.jscape.com	CSCOjaws	A library of sophisticated, pre-built components for assembling commercial-quality Java applications. The product is written entirely in Java and the look, feel and behavior is the same on all Java enabled environments.
JScrollPane 1.5.1	http://www.jscape.com	CSCOjpw	A pattern matching and search engine written entirely in Java. Provides full support for the PERL 5 regular expression syntax.
JSRS 1.0	Open Source http://www.ashleyit.com/rs/main.htm	CSCOxrts	JavaScript Remote Scripting (JSRS) permits data transfers and exchanges between a web application and a backend server without embedding a Java applet in the web page or refreshing the JSP.
log4cpp 0.2.5	Open Source (LGPL) http://log4cpp.sourceforge.net	CSCOxrts	Log For C++ (log4cpp) is a library of C++ classes for flexible logging to files, syslog, IDSA and other destinations. It is modeled after the Log4j Java library and follows its API closely.
log4j 1.1.3, 1.2.8	Apache http://jakarta.apache.org/log4j/	log4j	Log For Java (log4j) permits runtime logging from Java applications without modifying binary. Log statements remain in shipped code without incurring a heavy performance cost. Logging behavior is controlled using a configuration file.
LotusXSL 0.16.3	Lotus http://www.alphaworks.ibm.com/tech/LotusXSL	CSCOxsl	XSLT processor for transforming XML documents into HTML, text, or other XML document types. Predecessor of the Xalan XSLT processors.
Macromedia Jrun 2.3.3	Macromedia	TBD	TBD
Microsoft IPSecPol 1.22	Microsoft	TBD	TBD
mod_ssl 2.8.17	Apache http://www.modssl.org/	CSCOxrts	Provides strong cryptography for the Apache Web server. Follows the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols, and uses the Open Source SSL/TLS toolkit OpenSSL.
OpenSSL 0.9.7a	Open SSL Project http://www.openssl.org/	CSCOxrts	Open Source toolkit for implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols. Includes a full-strength general purpose cryptography library. Basis of mod_ssl.
Oromatcher 1.0	Oro Inc. http://www.savarese.org/oro/	CSCOxrts	A set of regular expression pattern matching and utility classes for Java.

CISCO CONFIDENTIAL**Table 1-2 CWCS Third-Party Tools (continued)**

Name	Vendor	Package	Description
Perl 5.005_02	O'Reilly http://www.perl.com/pub	CSCOperl	Perl is a high-level programming language derived from the C programming language and to a lesser extent from sed, awk, the UNIX shell, and at least a dozen other tools and languages. Perl's process, file, and text manipulation facilities make it well-suited for tasks involving quick prototyping, system utilities, software tools, system management tasks, database access, graphical programming, networking, and worldwide web programming.
PowerSearch 1.5.1	TBD	TBD	TBD
SNMP V3 Library	TBD	TBD	TBD
Struts Framework 1.0	Apache http://jakarta.apache.org/struts/	CSCOxrts	Provides an open-source MVD framework for building Java web applications. Basis of the Cisco User Interface Infrastructure (UII) used in CWCS.
Sybase ASA 9.0.0	Sybase http://www.sybase.com/products/anywhere/index.html	CSCODb	Sybase Adaptive Server Anywhere (ASA) is the CWCS embedded SQL database engine. It provides access from Java via JDBC, C/C++ via ODBC, and Perl via DBI. CWCS also provides enhancements for monitoring the database connection, integration with the Daemon Manager, and backup and restore.
Tibco Rendezvous 7.1.28	Tibco http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp	CSCOess	Supports the CWCS Event Services Software server-message bus. Highly efficient, scalable, with APIs for load balancing and fault tolerance.
Tomcat 4.1.33	Apache http://jakarta.apache.org/tomcat/	tomcat	Tomcat is the CWCS servlet container, used to implement the Java Servlet and JavaServer Pages technologies used throughout CWCS.
UCS-SNMP 4.2.2, NETPLUS 1.x.960.925	The NET-SNMP Project http://www.net-snmp.org/	CSCOxrts	A package of SNMP tools, including: An extensible agent and a notification receiver; an SNMP library; tools to request or set information from SNMP agents; tools to generate and handle SNMP traps; SNMP-related perl modules; versions of the UNIX netstat and df commands using SNMP; a Tk/perl mib browser.
Visibroker for Java 4.5.1; Visibroker for C++ 4.11	Borland http://www.borland.com/visibroker/	CSCOVorb	VisiBroker for Java and C++ provides essential software for developing, deploying, and managing robust, dynamic, and scalable distributed object applications. It is an implementation of the Common Object Request Broker Architecture (CORBA) from Object Management Group (OMG).
Xalan C++ 1.3	Apache http://xml.apache.org/xalan-c/index.html	xalan	An XSLT processor for transforming XML documents into HTML, text, or other XML document types. Written in C++. It works with the Xerces-C++ XML parser.

CISCO CONFIDENTIAL**Table 1-2** CWCS Third-Party Tools (continued)

Name	Vendor	Package	Description
Xalan Java 2.4.1.0	Apache http://xml.apache.org/xalan-j/	xalan	An XSLT processor for transforming XML documents into HTML, text, or other XML document types. It works with the the Xerces Java XML parser.
Xerces C++ 1.5.1	Apache http://xml.apache.org/xerces-c/index.html	xerces	A validating XML parser written in a portable subset of C++. Parses, generates, manipulates, and validates XML documents. It works with the Xalan C++ XSLT processor.
Xerces Java 1.4.4	Apache http://xml.apache.org/xerces-j/index.html	xerces	A validating XML parser written in Java. Parses, generates, manipulates, and validates XML documents. It works with the Xalan Java XSLT processor.
XML4J 2.0.11	IBM (http://www.alphaworks.ibm.com/formula/XML)	CSCOxml4j	A validating XML parser written in 100% pure Java.

Where to Find the CWCS SDK

The CWCS Software Development Kits, along with links to all SDK-related documentation and code , are available via links on the [CWCS 3.0 SDK Web Portal](#) hosted by Cisco Engineering & Manufacturing Connection Online (EMCO; the URL is http://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=2537).

The CWCS 3.0.5 SDK Web Portal is accessible to all Cisco staff working within Cisco networks. If you have never accessed an EMCO web portal before, you will be asked to register online for an account. There may be a short access delay while your registration is processed.

For Further Assistance

For assistance with running the software, reporting problems, or questions about the software, contact: cs-ch-leads@cisco.com.

**Note**

You can log bugs in the DDTs system, using `CSC.embu; product=CWCS; component=sdk`

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 2

FAQs and Programming Hints

Answers to frequently asked questions (FAQs) and programming hints are given into the following topics:

- [General Topics](#)
- [Daemon Manager](#)
- [Online Help](#)

General Topics

This topic addresses typical questions that do not apply to any particular CWCS component.

Q. How do I obtain the CWCS web protocol and port number from a JSP?

A. You can do this for an application running under Tomcat by querying the `md.properties` file located at `NMSROOT/lib/classpath/`, using the following code:

```
<%@ page import = "com.cisco.core.ccr.CCRInterface" %>
<%@ page import = "java.util.Properties" %>
<%@ page import = "java.io.FileInputStream" %>
<%
    ccr = new CCRInterface();
    String propFile = ccr.getCCRData("CMFSSLInfoFile", "Custom");
    FileInputStream fis = new FileInputStream(propFile);
    Properties prop = new Properties();
    prop.load(fis);
    String port = prop.getProperty ("PX_PORT");
    String protocol = prop.getProperty ("PX_PROTOCOL");
%>
```

Q. Can CWCS support applications that must manage devices using multiple community strings?

A. Yes. The `SnmpOnJava` library in CWCS contains a property and API calls that support multiple SNMP credentials for IP ranges. The API and guidelines on implementing it, see the [“Using CWCS SNMP Services”](#) section on page 24-3.

CISCO CONFIDENTIAL

- Q.** What do I need to know about casuser and upgrading?
- A.** If you are installing your application onto CiscoWorks for the first time, you do not need to worry about the casuser. If you are upgrading an existing application, you need to review the [“Understanding and Implementing the casuser” section on page 21-21](#).

Remember that, because of the strengthening of security, you can only use port 1741. If you have port 80 hard coded, you need to update your code.

- Q.** How is the CD structured and where can I locate information on it?
- A.** The CD comprises the following directories:
- CodeSamples
 - printdocs
 - sdkhelp
 - image_nt or image_sol
 - readme_nt.txt or readme_unix.txt
- Q.** Can I pre-compile JSPs with CWCS? If so, how can I do it? What are the advantages of doing so?
- A.** A Java Server Page is normally compiled into its implementation class the first time a user accesses the page. Pre-compiling JSPs improves their performance in first-time accesses, and also reveals any syntax errors they may contain. You can precompile JSPs in a couple of ways:
- During Build Time: You use ANT or JSPC scripts given with **Tomcast** to precompile the JSPs into Java files, then use javac to compile them into class files and place them in the working directories.
 - During Runtime/Install Time: You call each JSP with the pre-compiled option. The JSPServlet will interpret the request and compile each JSP into a class file.

CWCS pre-compiles JSP during build time, by creating the **Tomcast** runtime structure and using the JSPC script provided by Tomcat to compile into java files. It then changes the package of java files into “org.apache.jsp”, compiles them into class files, and makes them part of the protopackages. All these class files are copied into the working directory during installation.

The CWCS scripts are at /vob/ismg-core/install/image/extract/bin. The CWCS build script calls the main precompile script prejsp.sh, which in turn uses jspc.sh and prejsp-build.sch. These scripts are specific to Common Services, but you can follow the same approach using these scripts as a reference.

- Q.** What do I enter during installation and when running commands on Windows platforms? This book provides only UNIX references.
- A.** Enter C:\Progra~1\CSCOpX to reference the Windows platform directory. Do not spell out “Program Files”, since this will cause installation and command errors.
- Q.** When running Java server-side applications, should we use the cwjava command or directly reference the Java binaries?
- A.** Use the **cwjava** command. It provides a controlled runtime environment for CWCS-based Java server applications. For more information about cwjava, see the [“Launching a Java Application” section on page 4-2](#).

CISCO CONFIDENTIAL

- Q.** When I try running `cwjava`, I get the error message, Unable to locate CW2000 installation directory.
- A.** When running `cwjava` from the command line, you must provide the `-cw` parameter. For example:

```
cwjava -cw /opt/CSCOpX ...
cwjava -cw C:\Progra~1\CSCOpX\ ...
```



Note On Windows platforms, the install directory should be specified without spaces (as shown above).

- Q.** When SSL is enabled on the server, the HTTP call from a device is blocked. Why does this happen? How can I fix this?
- A.** When SSL is enabled on the CWCS WebServer, access to the webserver is through HTTPS only. This means the HTTP port is blocked when SSL is enabled on the server. When the server is running in SSL (https mode), any HTTP call will fail.

An API is provided to fix this problem. This API defines necessary rules in the webserver configuration file by reading the list of entries (identified case-sensitive) from a file (`addSetEnvIf.txt`) under `disk1` of the application's image. This API provides http access to the URLs of specified patterns.

SSL-complaint applications must identify scenarios where HTTP calls are made to the server, while running in the SSL enabled mode. The applications should do the following:

1. Identify the list of unique patterns in the URLs which should be accessible via HTTP when SSL is enabled.
2. If there are any such scenarios, create a file, `addSetEnvIf.txt` in `disk1`. This file should contain all such patterns.



Note During the application install, the install framework will define the required rules in the webserver configuration file based on the contents of `addSetEnvIf.txt`.

For example:

In RME's SWIM module, some devices (CSS11000 and NAM) contact the server through HTTP for image upgrading.

The format of the url would be `http://servername:1741/swimtemp/c6nam.2-1-2.bin.gz` These images reside under `/opt/CSCOpX/htdocs/swimtemp`. In SSL mode, the above url request fails since port 1741 (used for HTTP connections) is blocked. In this case you have to allow the URLs that contain `swimtemp`, for which the contents of the `addSetEnvIf.txt` should be: `swimtemp`

- Q.** I installed CWCS on my system, but I don't see any database engine service running as part of CWCS. What must I do to start this service?
- A.** Installing CWCS does not automatically start the CWCS database. The CWCS database engine is only running when the CWCS System Services bundle is enabled.

To enable the CWCS System Services bundle, see the [“Enabling New Service Bundles from the Command Line”](#) section on page 5-7 or [“Using CMFEnable”](#) section on page 5-9.

CISCO CONFIDENTIAL

- Q.** When I create my application, what methods should I use to report errors?
- A.** The method you use to report errors depends on where the error occurs:
- The UI component of an application should provide popup error dialog boxes.
 - Server components of an application should log errors to a log file. Log files are stored in the LOG directory (see [Chapter 3, “Understanding the CWCS Directory Structure”](#)).
- Q.** Are the application IDs passed in CAM authorization and registered with CCR case-sensitive?
- A.** Yes. Authorization will fail if the application ID in the CAM API doesn't match exactly with the application ID registered in CCR.
- Q.** How can I enable MICE debugging?
- A.** In the NMSROOT/MDC/tomcat/webapps/classic/WEB-INF/web.xml file, change the DEBUG parameter value to read as follows:

```
<context-param>
<param-name>DEBUG</param-name>
<param-value>true</param-value>
<description>MICE debug enabling</description>
</context-param>
```

Daemon Manager

This topic answers typical questions about the Daemon Manager (also known as Process Manager).

- Q.** On Windows, we can integrate our GUI server with the Daemon Manager. It runs, but the GUI does not appear. We registered Notepad as a test and that did not appear either.
- A.** The Daemon Manager's job is to run programs that are typically background processes, which do not have a user interface, but takes up a lot of computing resources. In fact, part of the command line that the Daemon Manager uses to start a process redirects the stdout and stderr outputs to log files. If your application has both user interface and backend processing combined into one binary, then you should not register it with the Daemon Manager. If the application is web-enabled (can run in a browser), then you can register it with the desktop.
- Q.** Can I register a Windows NT Service using pdreg?
- A.** Yes. In order to register the service *ServiceName* with the Daemon Manager, you have to run: **pdreg -r *ServiceName*** where *ServiceName* is the name used to register this service with NT Service Manager.

**Note**

The **pdreg** command allows you to register a process as a service, but you first have to create a Windows NT service (by registering it with Windows NT Service Manager) and then register the service with the Daemon Manager.

The Daemon Manager currently does not support Windows NT services completely: it will not restart the Windows NT Service if it fails. The Daemon Manager does that for normal processes. This may be a bug.

CISCO CONFIDENTIAL

- Q.** Where in my program should I define the “Process Name” defined in `pdreg` and `pdshow` ?
- A.** The process name you use to instantiate the Daemon Manager object is the process name that appears in `pdreg` and `pdshow` command. For example, in Java, if you use `dMgt theMgr = new dMgt (myDaemon) ;` then `myDaemon` is the process name used in `pdreg` and `pdshow`.
- Q.** Should I stop the Daemon Manager to register a new process?
- A.** No. You can register a process while the Daemon Manager is still running. However, the Daemon Manager will not start the execution of the newly-registered process. You must run the **pdexec** command to start this newly-registered process.
- Q.** If processes P2 and P3 depend on process P1, why do I have to run the `pdexec` for both P2 and P3 after P1 terminates?
- A.** After process P1 terminates, processes P2 and P3 will be brought down since they depend on P1. Daemon Manager will restart the crashed process P1. However, Daemon Manager does not keep the active process list when P1 terminates. Therefore, only P1 will be brought up again. P2 and P3 must be brought up manually using **pdexec**.
- Q.** If I use `kill -9` to kill a process, why is the process not restarted automatically by the Daemon Manager?
- A.** This happens because signal `-9` is the forced termination request issued by an operator.

Online Help

This topic answers typical questions about implementing online help.

- Q.** When the search engine displays the results, there is a blue column on the right that says `In:Unknown`. I would like this to state the name of my book. Is there a parameter I can set that will do this for me? I am implementing a client-only help system without any drop-ins.
- A.** Add the **SEARCH** line to your mapping file, as follows:


```
SEARCH, "YOUR APP NAME", "searchfile.sch"
```

 Then you should see *YOUR APP NAME* instead of `Unknown`.
- Q.** Must the **DROPINPLACE** lines always reference the `index.html` file? Even if I have more than one **DROPINPLACE** line in the mapping file? For example, I have three such lines in my help system’s mapfile:


```
DROPINPLACE, "1", "Server Configuration"
DROPINPLACE, "2", "Server Configuration", "What's New", /cmf/index.html
DROPINPLACE, "3", "Server Configuration", "Desktop", /cmf/index.html
```
- A.** Your **DROPINPLACE** lines can point to any file, even the same file. If you choose to have these lines point to *no* file, make sure the line is a folder, just like your first example. Otherwise nothing happens when a reader clicks on the node. For more information about mapping files, see the “[Understanding Mapping Files](#)” section on page 11-14.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 3

Understanding the CWCS Directory Structure

The following topics describe the CiscoWorks Common Services (CWCS) directory structure:

- [About CWCS Directory Policies](#)
- [About the CWCS Top-Level Runtime Directories](#)
- [About CWCS File Permissions](#)
- [About CWCS Property Files](#)
- [About CWCS Log Files](#)

About CWCS Directory Policies

Consider the following when creating runtime directories:

- Follow the guidelines defined in this chapter.
- Provide administrative options to relocate the folders that expand over time. Such folders will contain files that grow as new applications are added.
- Use the CWCS runtime structure explained in this chapter. This will allow your application to maintain compatibility with CWCS and other applications using CWCS.

About the CWCS Top-Level Runtime Directories

The CWCS top-level runtime directories are located in two trees that determine all other directory paths:

- *NMSROOT* contains the bulk of the product. This is the top-level install directory. The default *NMSROOT* definition differs for Windows and UNIX systems:
 - On Windows: *D:\Progra~1\CSCOpX*, where *D* is the drive letter stored in the Windows SystemDrive variable. CWCS uses the abbreviated DOS alias for Windows directory and file names. For example: *Progra~1* for the Program Files directory.
 - On UNIX: */opt/CSCOpX*.
- A directory that contains the logs and application-specific files created at run time. This is the top-level runtime files directory.

A system that ships simultaneously on both UNIX and Windows can have separate runtime trees for each platform, and these may look very different. Accordingly, the following topics describe both the common and platform-specific organization and contents of the top-level runtime directories:

CISCO CONFIDENTIAL

- [About the CWCS Common Directories](#)
- [About the CWCS Solaris-Specific Directories](#)
- [About the CWCS UNIX-Specific Directories](#)
- [About the CWCS Windows-Specific Directories](#)

About the CWCS Common Directories

The CWCS directories listed in [Table 3-1](#) are common to all platforms.

Table 3-1 Common Directories

Common Name	Directory path	Contents
BACKUP	<i>NMSROOT/backup</i>	Location for database and directory manifest files.
BIN	<i>NMSROOT/bin</i>	Program startup scripts - non-binary batch files, shared binaries, shared Windows DLLs.
CGI-BIN	<i>NMSROOT/cgi-bin</i>	cgi-bin 1.1 programs (web server can access this directory).
CLIENTINSTALL	<i>NMSROOT/cam_-repository</i>	Client-side install classes and control files.
CLIENT-JAVA	<i>NMSROOT/www/classpath</i>	Java class files for clients and those shared by client and server (web server can access this directory).
CLIENT-JAVA-APPS	<i>NMSROOT/www/classpath/com/cisco/nm/app_name</i>	Java classes for application/CWCS shared server/client code developed by Cisco. Application folders must contain all property files associated with the application.
COLLECT	<i>NMSROOT/collect</i>	Collect Server Info scripts and output files.
CONF	<i>NMSROOT/conf</i>	Configuration data files.
DATABASES	<i>NMSROOT/databases</i>	Original or empty database versions.
DBMS	<i>NMSROOT/objects/db</i>	Install directory for database engine.
DBUPGRADE	<i>NMSROOT/dbupdate</i>	Database upgrade files.
ETC	<i>NMSROOT/etc</i>	Copyright, readme, .profile, .cshrc, ... files.
HELP	<i>NMSROOT/htdocs/help</i>	Help file directory root.
HTDOCS	<i>NMSROOT/htdocs</i>	HTML tree used by the httpd server (web server can access this directory).
HTDOCS-IMAGES	<i>NMSROOT/htdocs/images</i>	Shared image files, application directories at this level for application specific images.
LIB	<i>NMSROOT/lib</i>	Shared libraries and directories of shared UNIX libraries (*.so).
MDC	<i>NMSROOT/MDC</i>	CWCS Core components.
NMIM	<i>NMSROOT/nmim</i>	Network Management Integration Utility Modules
OBJECTS	<i>NMSROOT/objects</i>	Install directory for non-Java and non-web server accessed CWCS/application code.
OBJECTS-APPS	<i>NMSROOT/objects/app_name</i>	CWCS and application install directories.
SELFTEST	<i>NMSROOT/selftest</i>	Product self-test Perl scripts.
SERVER-JAVA	<i>NMSROOT/lib/classpath</i>	Server-only Java classes.

CISCO CONFIDENTIAL**Table 3-1** Common Directories (continued)

Common Name	Directory path	Contents
SERVER-JAVA-APPS	<i>NMSROOT/lib/classpath/com/cisco/nm/app_name</i>	Java classes for application/CWCS server code developed by Cisco. Application folders must contain property files associated with this application unless otherwise defined in CLIENT-JAVA-APPS.
SETUP	<i>NMSROOT/setup</i>	Install information and results.

About the CWCS Solaris-Specific Directories

The CWCS directories listed in [Table 3-3](#) are unique to Solaris systems.

Table 3-2 Solaris-Specific Directories

Common Name	Directory path	Contents
DMCONFIG	/etc/rc.config.d	Solaris Daemon Manager config file.
DMSTARTUP	/etc/init.d	Solaris Daemon Manager startup files.

About the CWCS UNIX-Specific Directories

The CWCS directories listed in [Table 3-3](#) are unique to UNIX systems or typically not nested under the *NMSROOT* tree on UNIX. They are common to all UNIX systems, including Solaris.

Table 3-3 UNIX-Specific Directories

Common Name	Directory path	Contents
FILES	<i>User-specified/CSCOpX/files</i>	Product data files. The default for <i>User-specified</i> is /var/adm.
INSTLOGS	/var/tmp	UNIX install logs.
LOG	<i>User-specified/CSCOpX/log</i>	Log files generated by CWCS components and apps. The default for <i>User-specified</i> is /var/adm. Note that a few components and applications create logs in other directories. For details, see the “About CWCS Log Files” section on page 3-7.
MAN	<i>NMSROOT/man</i>	UNIX man files for product features.
TEMP	/tmp /temp	Default temporary directory.

About the CWCS Windows-Specific Directories

The CWCS directories listed in [Table 3-3](#) are unique to Windows or typically not nested under the *NMSROOT* tree on Windows.

Table 3-4 Windows-Specific Directories

Common Name	Directory path	Contents
FILES	<i>NMSROOT/files</i>	Enduser data.

CISCO CONFIDENTIAL**Table 3-4 Windows-Specific Directories**

INSTALL LOG	<i>system drive root</i> (e.g., C:\)	Windows install and uninstall log files. Filenames are of the form <i>Ciscoverks_setupxxx.log</i> , where <i>xxx</i> is a number indicating the serial order of the install or uninstall.
LOG	<i>NMSROOT</i> \log	Log files generated by CWCS components and applications. Note that a few components and applications create logs in other directories. For details, see the “ About CWCS Log Files ” section on page 3-7.
PROXY	<i>NMSROOT</i> \proxy	Proxy working directory.
TEMP	<i>NMSROOT</i> \temp	Default temporary directory.
TFTPBOOT	<i>NMSROOT</i> \tftpboot	TFTP directory.

About CWCS File Permissions

In general, the user *casuser* runs applications for Windows and UNIX platforms. Programs (binaries, scripts, etc) should be executable only by the user *casuser* in user group *casusers*. All files should be readable by users that belong to the group *casusers*.

- On UNIX platforms, set the permissions to:
 - Executables: `rwxr-x--- casuser casusers`
 - All normal data files: `rw-r----- casuser casusers`
 - Log files: `rw-rw---- casuser casusers`
 - Directories: `rw-rwxr-x casuser casusers`
- On Windows, set the access bits on the directories that contain the applications to allow execute and access by user groups *casusers* and *Administrators*.

**Note**

The user name *casuser* stands for “Cisco Application Server User”. Installation of CWCS on all platforms creates the group *casusers* and adds the user *casuser* to that group automatically.

About CWCS Property Files

Table 3-5 lists all the CWCS property files, sorted alphabetically by the components they configure, with their locations. All paths are relative to the top-level *NMSROOT* directory on each platform (see the “[About the CWCS Top-Level Runtime Directories](#)” section on page 3-1).

CISCO CONFIDENTIAL**Table 3-5 CWCS Property Files**

Component	Directory Path(s)	File(s)	Description
CiscoWorks	/lib/classpath/	apps-plugin.properties, backupstatus.properties, cam.properties, javaplugin.properties, jrmuser.properties, md.properties, proxy.properties, ss.properties; ssl.properties, sso.properties	CiscoWorks configurations
	/www/classpath/com/cisco/nm/cmfmf/	debug.properties	Sets debug levels for MakerChecker, other application modules
CiscoWorks Home Page	/lib/classpath/com/cisco/nm/cmfservlet/	DesktopServlet.props	Sets the URL for the Desktop messaging banner
	/MDC/tomcat/webapps/cwhp/WEB-INF/resources	cwhpadmin.properties	CWHP Admin configuration
CMIC	/conf/cmif/	log-cmic.properties	Log4J logging configuration for this component
Common Services Home	/lib/classpath/	CSHP.properties, log4j-cshp.properties, setup-log4j.properties, setup.properties,	Common Services Home configuration
Database Services	/www/classpath/com/cisco/nm/cmfdbservice	/DBServer.properties, /orig/DBServer.properties, /servlet/CommServlet.properties	CWCS Database configuration
	/lib/classpath/iAnywhere/ml/jdbcodbc/	iresource.properties (and national language variants such as iresource_de.properties, iresource_en.properties and iresource_fr.properties)	Sybase SQL Anywhere configuration
DCR	/lib/classpath/com/cisco/nm/dcr/	log4j-dcr.properties, log4j-dcrclient.properties	Log4J logging configuration
	/MDC/tomcat/webapps/cwhp/WEB-INF/classes/	DCR_Implementation_Details.properties	System configuration
	/objects/dcrimpexp/Adaptors	/HPOV6.x/Adaptor.properties, /Netview7.x/Adaptor.properties	Import/Export adaptor configurations
	/objects/dcrimpexp/cnf	DCRImpExp.properties	Import/Export process parameters
	/objects/dcrimpexp/conf/	log-dcr.properties	Log4J logging configuration for this component

CISCO CONFIDENTIAL**Table 3-5 CWCS Property Files (continued)**

Component	Directory Path(s)	File(s)	Description
Device Center	/conf/devicecenter/	log-devicecenter.properties	Device Center logging
	/lib/classpath/com/cisco/nm/pidm/	log4j-pidm.properties	Log4J logging configuration for this component
	/lib/classpath/com/cisco/nm/pidm	PIDM_Registration_data	Configuration data used for PIDM bypass in Device Center
Device Selector	/MDC/tomcat/webapps/cwhp/WEB-INF/classes/	DeviceSelector.properties	Device Selector configuration
	/MDC/tomcat/webapps/cwhp/WEB-INF/classes/	log4j-devsel.properties	Log4J logging configuration for Device Selector
EDS	/www/classpath/com/cisco/nm/cm/eds/	display/realtimeDisplay.properties, error/EDS_Error_Messages.properties gcf/gcf.properties trap/trapReceiver.properties ui/helpSupport.properties	EDS real-time display, error messaging, trap support, GCF, UI configuration.
ESS	/NMSROOT/objects/ess/conf	essproperties.conf	ESS configuration
Javac	/www/classpath/sun/tools/javac/resources/	javac.properties	Javac configuration
Job Scheduler	/www/classpath/com/cisco/nm/cm/scheduler/	JobRegistration.properties, JobScheduler.properties	Job registration and scheduling configuration
JRE 1.4.1	/lib/jre/lib/	content-types.properties, flavormap.properties, font.properties (per language) , logging.properties, psfontj2d.properties	JRE 1.4.1 content types, flavormap, font properties, logging
	/setup/, /setup/dependency/	Various property files	JRE 1.4.1 setup dependencies
Log4J	/objects/licenses/	log4j-license.properties	Log4J license
	/MDC/tomcat/webapps/cwhp/WEB-INF/classes/	log4j.properties, log4j.server.properties	Log4j server configuration
MDC	/MDC/Sybase/Shared/Sun/jdk122/jre/lib/, /MDC/Sybase/Shared/Sun/jdk122/jre/lib/images/cursors/	Various	JDK 1.2.2 content type, flavormap, font and cursor support for MDC Sybase
	/MDC/JRE/lib/	Various	JRE 2content type, flavormap, font and cursor support configuration for MDC
	/MDC/tomcat/conf/jk/	workers.properties	Tomcat configurationfor MDC

CISCO CONFIDENTIAL**Table 3-5 CWCS Property Files (continued)**

Component	Directory Path(s)	File(s)	Description
OGS	/MDC/tomcat/webapps/cwhp/WEB-INF/classes/	log4j-ogs.properties	OGS logging properties for log4j
	/MDC/tomcat/webapps/cwhp/WEB-INF/classes/	OGSClient.properties, OGSServer.properties, SharedGroups.properties	OGS runtime configuration
	/MDC/tomcat/webapps/cwhp/WEB-INF/resources	OgsProviderGroup.properties	Provider Group Name change feature configuration
	/lib/classpath/com/cisco/nm/cmfdbi/	dbi.properties, deldev.properties	OGS group structure, device deletion
PSU	/lib/classpath/com/cisco/nm/xm/psu/conf/	log4j-psu.properties, psu.properties, tag.properties	PSU runtime and logging properties
Security	/www/classpath/com/cisco/nm/cm/security/jaas/	Twofish.properties	TwoFish encryption configuration
Visibroker	/etc/	Orb.properties	Basic Visibroker ORBconfig
	/lib/jre141/lib/endorsed/com/inprise/vbroker (and subdirectories)	Various	Security, events, HIOP, messages, names support for Visibroker
	/lib/jre141/lib/images/cursors/	Various	JRE 1.4.1 cursor support for Visibroker
	/www/classpath/com/inprise/vbroker/	Various	Visibroker support
Web Services	/MDC/tomcat/webapps/cwhp/WEB-INF/resources/work/	modules.properties <could not find this file in the directory structure>	Tomcat worker configuration
<These property files are not documented above>	/conf	backup.properties, log4j-smtp.properties, /cam/acsmmap.properties	
	/lib/classpath/com/cisco/nm/util	cstm-registry-view.properties	
	/lib/classpath/com/cisco/nm/xm/s/vds	vds.properties	
	/MDC/tomcat/conf	shared.properties	
	/MDC/tomcat/webapps/cwhp/WEB-INF/classes	trans.properties	

About CWCS Log Files

Table 3-6 lists the filenames and locations of all logs produced by CWCS components.

The table refers to normal top-level log directories for each platform. These are *NMSROOT*/log on UNIX and *NMSROOT*\log on Windows (see the “[About the CWCS Top-Level Runtime Directories](#)” section on page 3-1). Nearly all CWCS components write their activity logs there. Paths to log files other than those in the normal log directories are shown in the table relative to *NMSROOT* on each platform (e.g.: The path to license.log is \Program Files\CSCOpX on Windows, /var/opt/CSCOpX/ on Solaris.).

CISCO CONFIDENTIAL**Table 3-6 CWCS Log Files**

Component /Module	Directory Path	File	Description
Backup and Restore	/MDC/tomcat/vms/maas/Data	CLIBackup.log	Core-based backup logs (command-line interface)
	/MDC/etc/backup-restore/debug-log	BackupServerDebugOut.txt, RestoreServerDebugOut.txt	Core-based backup and restore debug logs
	Normal top-level log directories	dbbackup.log, restorebackup.log	CMF-based backup and restore logs
Core Admin Module (CAM)	/MDC/log/	core*	Logs for Authentication, Authorization and Accounting process
CRMLogger	Normal top-level Windows log directory only	sys log.log	Syslogs received from device/machine. Windows only.
	Normal top-level Windows log directory only	syslog_debug.log	CRMLogger debugging information and messages from device/machine. Windows only.
CWCS General	Normal top-level log directories	error.log	General CWCS errors
	Normal top-level log directories	perlerr.log	Perl interpreter errors
	Normal top-level log directories	Proxy.log	Proxy activity
	Normal top-level log directories	event.log	CWCS events
	Normal top-level Solaris log directory only	daemons.log	All Daemon Manager-controlled processes. Solaris only.
Database Services	/databases/cmf/	cmf.log	Database access.
	Normal top-level log directories	CmfDbMonitor.log	Sybase database operations
	Normal top-level log directories	dbpwdChange.log	Database password changes
	Normal top-level log directories	dmgtDbg.log	Daemon Manager interactions with Sybase database
	/objects/db/win32/	dbcond8.log	Database condition log
DCR	Normal top-level log directories	dcr.log	DCR log
DCR Import Export	Normal top-level log directories	dcrimpexp.log, DCRServer.log (Windows Only), daemons.log (Solaris Only)	
DiskWatcher	Normal top-level log directories	diskWatcher.log	DiskWatcher warnings
EDS	Normal top-level log directories	EDS-GCF.log, EDS.log	All EDS activity
ESS	Normal top-level log directories	ESS.log, JavaDebug.log	All ESS activity

CISCO CONFIDENTIAL**Table 3-6** CWCS Log Files (continued)

Component /Module	Directory Path	File	Description
JRM	Normal top-level Windows log directory only	jrm.log	JRM server activity on Windows only
Licensing APIs	Normal top-level log directories	LicenseServer.log	License Server activity
	Normal top-level log directories	license.log	Product license changes
LWMS	Normal top-level log directories	lwms.log	Lightweight Messaging Service activity
OGS	Normal top-level log directories	OGSClient.log CMFOGSCient.log	OGS client module log
	Normal top-level log directories	CMFOGSServer.log	OGSServer log (CWCS OGS Server only)
PSU	/lib/classpath/com/cisco/nm/xms/psu/conf/	psu.log	All PSU activity
Visibroker	Normal top-level Windows log directory only	RmeGatekeeper.log	RME Gatekeeper activity from Visibroker vorb package. Windows only
Web Services	/MDC/apache/logs/	access.log, error.log, mod_jk.log, ssl.log	All Apache activity
	/MDC/tomcat/logs/	jasper-YYYYMMDD.log, servlet-YYYYMMDD.log, stderr.log, stdout.log,	All Tomcat activities
	Normal top-level log directories	changeport.log	Port change information
<List of other logs that are not documented>		dbupdate.log CSRegistryServer.log dcmaservice.log deviceselector.log EDS-TR.log TomCatMonitor.log	

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 4

Understanding the CWCS Execution Environment

The following topics describe the CWCS execution environment and provide guidelines for creating applications to run in this environment:

- [Understanding the Java Application Launch Process](#)
- [Launching a Java Application](#)
- [Using Servlets and JSPs with Tomcat](#)
- [Using JavaBeans](#)



Note

As of this release, CWCS has dropped all support for Macromedia JRun/JSP. JRun is no longer included in any version of the CWCS distribution.

Understanding the Java Application Launch Process

The CWCS Java runtime environment is similar to the standard Java2 environment, with the following exceptions:

- The bootstrap class path contains a Visibroker ORB implementation followed by standard JRE2 classes. This ensures that the VisBroker classes will be used to implement ORB.
- The `com.cisco.nm.cw2000.home` system property is defined and points to the location of the CWCS installation root.
- The default user class path contains the CWCS server class hierarchy (`lib/classpath`) and the CWCS client class hierarchy (`www/classpath`).
- The current working directory is set to the CWCS installation root.
- On UNIX platforms, the path for JNI code (the value of the `LD_LIBRARY_PATH` variable for Solaris, and so on) contains the CWCS shared objects directory library.
- On Windows platforms, it is assumed that `cwjava` resides in the same directory as the CWCS JNI DLLs.
- CWCS supports two JRE versions on the server side: **1.3.1_06** and **1.4.2_10**. Version **1.4.2_10** is available under `NMSROOT/lib/jre`. Version **1.3.1_06** is available under `NMSROOT/MDC/jre`.

CISCO CONFIDENTIAL

When `cwjava` launches a Java application in the CWCS runtime environment, it performs these functions:

1. Sets `CW2000` to the CWCS installation directory. The default value is the value of the `NMSROOT` environment variable. To override the default, use the `-cw` option (see the “[Launching a Java Application](#)” section on page 4-2).
2. Sets `JRE` to the Java Runtime Environment root directory. By default, it is `NMSROOT/MDC/jre` (which means `cwjava` uses JRE 1.3.1_06 by default). To override the default, use the `-cw:jre` option.
3. Sets `VB` to the VisiBroker ORB path. By default, the path is
`CW2000/www/classpath/vbjapp.jar:CW2000/www/classpath/vbjorb.jar`.
 To override the default, use the `-cw:vb` option.
4. Sets the value of `com.cisco.nm.cw2000.home` system property to `CW2000`.
5. Sets the current directory to the value of the `-wd` option. If `-wd` is omitted, it sets the current directory to `CW2000`.
6. Processes the rest of the options as standard Java and launches the interpreter.
7. The following ORB options are automatically added by `cwjava`. You need not specify these in the command line.
 - `-Dorg.omg.CORBA.ORBClass=com.inprise.vbroker.orb.ORB`
 - `-Dorg.omg.CORBA.ORBSingletonClass=com.inprise.vbroker.orb.ORBSingleton`
 - `-Djavax.rmi.CORBA.StubClass=com.inprise.vbroker.rmi.CORBA.StubImpl`
 - `-Djavax.rmi.CORBA.UtilClass=com.inprise.vbroker.rmi.CORBA.UtilImpl`
 - `-Djavax.rmi.PortableRemoteObjectClass=com.inprise.vbroker.rmi.PortableRemoteObjectImpl`

Launching a Java Application

Use this syntax to launch a Java application:

```
cwjava [ options ] class [ argument ...]
```

```
cwjava [ options ] -jar file.jar [argument ...]
```

If the `-jar` option is specified, the first non-option argument is the name of the JAR archive containing class and resource files for the application. The startup class is indicated by the `Main-Class` manifest header.

CISCO CONFIDENTIAL

Table 4-1 summarizes the command line options that modify the runtime environment.

Table 4-1 *cwjava Command Line Options*

Option	Description
-cw <i>dir</i>	Sets the CiscoWorks installation directory to <i>dir</i> .
-cw:jre <i>dir</i>	Sets the JRE directory root to <i>dir</i> . This directory should contain the standard hierarchy of JRE files—the interpreter in the <i>dir/bin</i> and standard classes in the <i>dir/lib</i> . If the path name is not absolute, it is relative to the CWCS Server installation directory. For example: <code>cwjava -cw:jre NMSROOT/MDC/jre</code> .
-cw:vb <i>vbfile</i>	The path containing VisiBroker library. Elements can be directories or JAR files. All non-absolute elements are relative to the CWCS Server installation directory.
-cw:nojit -cw:jit	Disables/enables the just-in-time (JIT) compiler. Unless the -Xdebug option is present, the JIT compiler is enabled by default.
-cw:xrs	This new option is applicable on Windows platforms only. It forces CWCS to ignore the CTRL_LOGOFF_EVENT <i>only</i> . Use this option if: <ul style="list-style-type: none"> Your application specifies JRE 1.3 or above, via the -cw:jre option. The application runs as a Windows service. The service aborts when a Windows user on the CiscoWorks Server logs off from the machine. This option is a subset of the Java -Xrs option in its current definition. Use it instead of the Java -Xrs option. If you use Java -Xrs , it will ignore other events and signals in addition to CTRL_LOGOFF_EVENT. This will cause Ctrl-Break thread dumps to be unavailable, and will force your code to be responsible for causing shutdown hooks to run.
-wd <i>workingdir</i>	Sets the current directory to <i>workingdir</i> before launching the interpreter. All non-absolute elements are relative to the CWCS Server installation directory.
-cp:a <i>classpath</i> -cp:p <i>classpath</i>	Prepends/appends classpath to the class path. All non-absolute elements are relative to the CWCS Server installation directory. These two options cannot be specified at the same time.
-cp:amf <i>manifest_file</i> -cp:pmf <i>manifest_file</i>	Prepends/appends Manifest file containing directories, zip or jar files to the class path. All non-absolute elements are relative to the CWCS Server installation directory. These two options cannot be specified at the same time.
-classpath <i>classpath</i> -cp <i>classpath</i>	Sets class path to <i>classpath</i> . All non-absolute elements are relative to the CWCS Server installation directory.
-? -help	Displays help information and exits.
-version	Displays version information and exits.

CISCO CONFIDENTIAL

Table 4-2 summarizes the options that `cwjava` processes in the same manner as the Java tool from JRE2.

Table 4-2 *cwjava* JRE2 Options

Option	Description
<code>-Dproperty=value</code>	Sets a system property value.
<code>-jar</code>	<p>Runs a program encapsulated in a JAR file. The first argument is the name of a 'JAR file instead of a startup class name. For this option to work, the manifest of the JAR file must contain a line of the form <code>Main-Class: classname</code>. Here, <code>classname</code> identifies the class that contains this method, which serves as your application's starting point:</p> <pre>public static void main(String[] args)</pre> <p>When you use this option, the JAR file is the source of all user classes, and other user class path settings are ignored.</p>
<code>-verbose</code> <code>-verbose:class</code>	Displays information about each class that is loaded.
<code>-verbose:gc</code>	Reports on each garbage collection event.
<code>-verbose:jni</code>	Reports information about the use of native methods and other Java Native Interface activity.
<code>-Xbootclasspath:bootclasspath</code>	Specifies a list of directories, JAR archives, and ZIP archives to search for boot class files. These are used in place of the boot class files included in the JDK 1.2 software. <i>Use with care.</i>
<code>-Xdebug</code>	Starts with the debugger enabled. This option automatically disables JIT. The Java interpreter prints out a password for <code>jdb</code> 's use.
<code>-Xnoclassgc</code>	Disables class garbage collection.
<code>-Xmsn</code>	<p>Specifies the initial size of the memory allocation pool. This value must greater than 1000:</p> <ul style="list-style-type: none"> To multiply the value by 1000, append the letter <code>k</code>. To multiply the value by 1 million, append the letter <code>m</code>. <p>The default value is <code>1m</code>.</p>
<code>-Xmxn</code>	<p>Specifies the maximum size of the memory allocation pool. This value must greater than 1000:</p> <ul style="list-style-type: none"> To multiply the value by 1000, append the letter <code>k</code>. To multiply the value by 1 million, append the letter <code>m</code>. <p>The default value is <code>64m</code>.</p>
<code>-Xrunhprof{ :help :suboption=value,...}</code>	<p>Enables CPU, heap, or monitor profiling. This option is typically followed by a list of comma-separated <code>suboption=value</code> pairs. Run this command to obtain a list of suboptions and their default values:</p> <pre>cwjava -Xrunhprof:help</pre>
<code>-Xrs</code>	Reduces the use of operating system signals.
<code>-Xcheck:jni</code>	Performs additional checks for Java Native Interface (JNI) functions.
<code>-X</code>	Prints help on java non-standard options

CISCO CONFIDENTIAL

Using Servlets and JSPs with Tomcat

This version of CWCS ships with Tomcat 4.1.29. This version of Tomcat contains a reference implementation of the Servlet 2.3 and JSP 1.2 specifications. To execute your Servlets and JSP under the Tomcat Servlet Engine, please observe the following guidelines.

- There are no special procedures needed to get Tomcat enabled and working with Apache. CWCS installs and configures Tomcat and Apache for you.
- Remember to include the package statement in the servlet code.
- Before executing your servlets and JSPs, you must register your Servlet Context with the Tomcat Servlet Engine. CWCS provides the UpdateTomcatMain API to register your servlet context with Tomcat. During installation, call this API in your package's post-install code. Calls to the registration API have the following form:

```
UpdateTomcatMain(UrlPattern, RelPath, Mapping, AppID, IsUninstall);
```

Where:

- *UrlPattern* is the name of your Servlet Context (for example: "MDC/Servlet"). To refer to "SampleServlet" in this context, you would call
`http://server-name:port/MDC/Servlet/SampleServlet`
- *RelPath* is the document base for this Servlet Context. The path should be a relative path with respect to the *NMSROOT*/MDC/tomcat directory (for example: "MDC/maas/servlet").
- *Mapping* is normally an empty string (for example: "").
- *AppID* is the name of the application using this context (for example: "core").
- *IsUninstall* is a flag indicating whether to add or remove the servlet context. To add the servlet context, set the value to "false". To remove it, set the value to "true".

For example:

To register a typical Servlet Context on Windows:

```
UpdateTomcatMain("/MAAS/JSP", "vms/maas/JSP", "\\\"\\\"", "maas-jsp", "false");
```

To do the same thing on Solaris:

```
UpdateTomcatMain "/MAAS/JSP" "vms/maas/JSP" $EMPTY_STRING "maas-jsp" "false" ;
```

Using JavaBeans

For JavaBean classes used in a JSP file that runs within CWCS, follow these guidelines::

- Place your JSP files in your webapp directory. The webapp directory needs to be specified while adding your Servlet Context. The directory is the same as the *RelPath* value specified when you call the UpdateTomcatMain API during installation (see the [“Using Servlets and JSPs with Tomcat” section on page 4-5](#)). The *RelPath* value is a relative path; it must be converted into an absolute path while placing the JSP.
- If your application is using the Struts framework or the User Interface Infrastructure (UII), then JSP files must be placed according to the framework specification.

CISCO CONFIDENTIAL

- Place your JavaBean class file under the \$WebAppDir/WEB-INF/classes directory. If your JavaBean is in a JAR file, place it under the \$WebAppDir/WEB-INF/lib directory.
- In `jsp:usebean`, include the fully qualified classname. For example:

```
<jsp:useBean id= 'myBean' class='com.cisco.nm.example.Testbean' scope='page' />
```



CISCO CONFIDENTIAL

CHAPTER 5

Getting Started with CWCS

CWCS is a collection of subsystems, execution environments, engines, and shared code libraries. This collection represents a software platform that provides services to web-based network management applications. CWCS allows you to:

- Create custom applications and add them to the desktop navigation tree.
- Use the CWCS backend services to include additional functionality in your application.
- Use the CWCS support services to manage your build environment or improve application startup time.

The following topics provide basic information on how to integrate your applications with CWCS:

- [How CWCS Works](#)
- [Installing CWCS](#)
- [Enabling CWCS Services](#)
- [Interacting with CWCS](#)

How CWCS Works

When you install your application into CWCS, your application:

1. Integrates with the CiscoWorks Home Page (CWHP). CWHP provides:
 - User authentication by means of a single user login.
 - Launch points for tasks, tools and administrative interfaces, including other CWCS-based and third-party applications.
 - Navigation between components in a suite of applications.
 - Registration of third-party applications not built with CWCS.



Note CWHP can be accessed using both Netscape and Microsoft Internet Explorer browsers.

2. Enables any necessary CWCS service layers (system, network, and core layers). For more information about enabling CWCS service bundles, see the [“Enabling CWCS Services” section on page 5-3](#).

When you enable the system layer, remember that you are enabling only a specific service subset. The only elements of the system layer that are documented in the SDK are the JRM and the database engine.

CISCO CONFIDENTIAL

3. Integrates with the CWCS Daemon Manager (also known as the Process Manager). Applications and CWCS backend services rely on the Daemon Manager for:
 - Initial startup following server boot or manual startup.
 - Restarting after a failure.
 - Starting only when the necessary prerequisite programs are up and running.

At runtime, your application can:

- Perform security verifications.
- Use other CWCS services as needed.

All interactions between CWHP and application services pass through the web server. Typically, clients interact with the server using a URL, which points to a servlet or web application, or a static web page. To add your application to CWCS, see the [“Integrating Your Application with CWHP” section on page 7-6](#).

Installing CWCS

To start working with the CWCS code, you must install Common Services 3.0.5. The prerequisite for Common Services 3.0.5 is Common Services 3.0.3.

Use the CiscoWorks Common Services 3.0.3 (Includes CiscoView) Installation CD (commonly known as “CD One”). This CD contains the base Common Services software, CiscoView and the Integration Utility. This CD is commonly known as the CWCS “CD One” edition because it replaces the previous CD One, which included CMF and CiscoView CWCS installation disk. All CiscoWorks applications require the installation.

When you install, you must select one or more of the following three installation options:

1. CiscoWorks Common Services (CWCS): Installs CWCS, which including md, perl, xml4j, hlp, xsl, web, tomcat, xrts, cam, jext, jcht, snmp, jgl and jre2.
2. CiscoView: Installs both CiscoView and Common Services.
3. Integration Utility: Installs the Integration Utility (formerly known as the Network Management Integration Module, or NMIM).

Due to software dependencies, you must install option 1 if you install option 2. This selection is enforced on Windows platforms automatically. On Solaris platforms, you must select each option manually.

After you have installed Common Services 3.0.3, mount the LMS 2.6 CD-ROM to install Common Services 3.0.5. The LMS 2.6 CD-ROM has the software updates for all applications in the LMS Bundle.

The version of CWCS components included in SDK downloads are intended for developers who want to create new CWCS-based applications. The downloads usually include only the CWCS base software, plus source code (per-product SDK downloads will contain only the software and source for that per-product component).

Related Topics

- See CWCS Installation Guides:
 - *Installation and Setup Guide for CiscoWorks Common Services 3.0.5 (includes CiscoView) on Solaris.*
 - *Installation and Setup Guide for CiscoWorks Common Services 3.0.5 (includes CiscoView) on Windows.*

CISCO CONFIDENTIAL

- See [Chapter 21](#), “Using the Installation Framework.”

Enabling CWCS Services

The following topics describe CWCS service bundles and how to use them to access various CWCS services:

- [Understanding CWCS Service Bundles](#)
- [Using CWCS Service Bundles](#)
- [Registering for CWCS Services](#)
- [Enabling New Service Bundles from the Command Line](#)
- [Using CMFEnable](#)

Understanding CWCS Service Bundles

CWCS services are grouped into the following service bundles:

- **CWCS Base Services:** Basic CWCS components necessary to support a web-based application. These components include the web server, CWCS security, the servlet engine, and JRE.
- **CWCS System Services:** CWCS components that add services such as JRM and the database engine.
- **CWCS Network Services:** CWCS components that enable ANI discovery and other network services.

**Note**

The Network Services layer is provided for backward compatibility with older applications that require network services. **ANI has been deprecated and is not included in CWCS 3.0.** Next Generation Discovery is available on a per-product basis as an ANI replacement. For information, see the *SDK Developer's Guide for Next Generation Discovery 1.0*, [EDCS-368448](#).

- **CWCS Core Services:** CWCS components that enable VPN and security administration and the TIBCO events engine.

At installation, the installation framework makes a call to the CWCS service bundles enabling mechanism, CMFEnable (see the [“Using CMFEnable”](#) section on page 5-5). This call provides CMFEnable with the list of CWCS service bundles required by the application suites that are being installed. Note that these service bundles exist to support an always-installed/running-as-needed service model. Applications that do not require all services can avoid wasting CPU cycles on them.

Using CWCS Service Bundles

The following are some tips you can use while working with CWCS services:

- The CWCS Base Services are enabled by default; no explicit request for this bundle is required.
- Network service bundles include System service bundles. If you enable Network services, all System service bundles are also enabled.

CISCO CONFIDENTIAL

- If your application needs a system or network service, then you must register the required service layer. For example, if your application needs the Sybase engine, then your application will need to register the system services layer of CWCS. This layer may contain many other services that you do not need, but you must register that layer to get any services from that layer (see the [“Registering for CWCS Services”](#) section on page 5-4).
- Application suites that have not registered for a CWCS service bundle at install time can use the CWCS service bundles enabling mechanism, CMFEnable, to enable CWCS service bundles (see the [“Enabling New Service Bundles from the Command Line”](#) section on page 5-5).
- When a bundle is enabled, it remains enabled until
 - A bundle or suite that enabled those CWCS service bundles is uninstalled.
 - A service bundle is manually disabled using CMFEnable.pl
- Note that installation of other products can reset the service registrations back to those registered for a suite of applications installed on the service.

Registering for CWCS Services

At installation, use the packaging information file to register for CWCS service bundles.

To register for a service bundle at installation:

-
- Step 1** Edit the `tag.pkgpr` file in the `/install` directory. This file is described in the [“Getting Started with the Installation Framework”](#) section on page 21-4.
- Step 2** Add a line that contains the `CMFSERVICES` keyword. This keyword has the valid token values `System`, `Network`, and `Core`. For example:
- The Campus Manager file includes the line:


```
CMFSERVICES=Network, System
```

Multiple tokens must be separated by a comma; spaces are not allowed.
 - The SLM file includes the line:


```
CMFSERVICES=System
```
 - If you are registering an MC application that requires explicit registration of Core services, the file includes the line:


```
CMFSERVICES=Core
```



Note The following MC applications do not require explicit registration of the Core service layer: PIXMC, QPM, IOSMDC, Router MC, IDSCFG, IDSMON, and AutoUpdate. If one of these applications is installed and registered with the Core Client Registry, the `CMFSERVICES` line will include a `Core` value.

Related Topics

- [Understanding CWCS Service Bundles, page 5-3](#)
- [Adding Your Application to the CiscoWorks Home Page, page 5-8](#)

CISCO CONFIDENTIAL

Enabling New Service Bundles from the Command Line

Application suites that have not registered for a CWCS service bundle at the time of installation must shut down both CWCS and all application services before enabling new CWCS services.

For example, to access the CWCS database engine service, you must enable the CWCS System Services bundle. To start the System Services manually:

Step 1 Stop the Daemon Manager:

- On a Solaris platform, enter:

```
/etc/init.d/dmgttd stop
```
- On a Windows 2000 or Windows NT platform, enter:

```
net stop crmdmgttd
```

Step 2 Start the CWCS System Services bundle:

```
NMSROOT/bin/perl NMSROOT/conf/cmfb/bin/CMFEnable.pl -FORCE system
```

where *NMSROOT* is the directory in which the product was installed.

Step 3 Start the Daemon Manager:

- On a Solaris platform, enter:

```
/etc/init.d/dmgttd start
```
- On a Windows platform, enter:

```
net start crmdmgttd
```

Step 4 In this example, to be sure that you have access to the CWCS database engine service, verify that the *CmfDbEngine* and *CmfDbMonitor* processes are running:

```
NMSROOT/bin/pdshow
```

where *NMSROOT* is the directory in which the product was installed.

**Caution**

If you start system services, you also disable network services. Also note that enabled service bundles may be reset if you install a new product on the server.

Related Topics

- [Understanding CWCS Service Bundles, page 5-3](#)
- [Using CMFEnable, page 5-5](#)

Using CMFEnable

The *CMFEnable* Perl script enables CWCS service bundles. This script is used at install time and is also available from the command line. For more information about services bundles, see the [“Understanding CWCS Service Bundles”](#) section on page 5-3.

CISCO CONFIDENTIAL

If CMFEnable finds that the Daemon Manager is still active, it displays a warning message which contains the command the user must enter to shutdown the Daemon Manager. CMFEnable aborts after displaying this warning message.



Caution

If you start system services, you also disable network services.

Name	CMFEnable.pl	
Description	Enables CMF service bundles	
Syntax	<i>NMSROOT</i> /lib/perl/install <i>NMSROOT</i> /conf/cmf/bin/CMFEnable.pl {-FORCE -INSTALL -CURRENT} [arg1]	
Syntax Arguments	Name	Description
	force	Enables a service bundle without verifying the system configuration.
	install	Forecasts the packages to be installed. This information is used by the installation facility.
	current	Enables and disables service bundles based on the current system configuration.
	arg1	<ul style="list-style-type: none"> • <i>arg1</i> is a token of comma-separated service bundle names. • With -force, if <i>arg1</i> is missing, all services are disabled. • Valid service bundle names are Network and System. • No spaces are allowed in the token.
Output	0	Success
	Non-zero	Failure. Values >0 may contain other interesting information depending on the routine.
Examples	<ul style="list-style-type: none"> • To have ANI running, but not install the Campus Manager software to get this functionality, use the following command to enable ANI services: <code><i>\${NMSROOT}</i>/conf/cmf/bin/CMFEnable.pl -INSTALL Network</code> • To enable the database without ANI, use the following command to enable System services: <code><i>\${NMSROOT}</i>/conf/cmf/bin/CMFEnable.pl -INSTALL System</code> 	

For example, consider this installation model:

-
- Step 1** Install CWCS.
 - Step 2** Then install Essentials.
 - Step 3** Then other applications.
-

CISCO CONFIDENTIAL

The INSTALL option is used to enable the service bundles before the Essentials packages are copied to the system.

This CURRENT option is called by the installation framework after all packages are installed on the system. The *.info files for all packages are examined to determine the required service bundles (System or Network), and the corresponding action (enable or disable) is performed. For example:

- If Network is required and the previous setup is System, then Network is enabled.
- If none is required and the previous setup is System, then System is disabled.

Related Topics

[Specifying Package Properties, page 21-6](#)

Interacting with CWCS

The following topics describe the ways you and your application can interact with CWCS:

- [Designing the User Interface](#)
- [Adding Your Application to the CiscoWorks Home Page](#)
- [Using the Backend Services on the CWCS Server](#)
- [Using the CWCS Support Tools](#)
- [Getting Up to Speed Quickly](#)

Designing the User Interface

Your application interface must present a consistent look and feel to the user. To help you achieve that goal, CWCS supports the User Interface Infrastructure (UII). The CWCS team recommends that you use the latest release of the User Interface Infrastructure (UII) to build your application interface. Version 6.1 and later of this UII are compatible with CWCS 3.0, and are available for download at <http://picasso>.

If you cannot use the UII, be aware that:

- Many CWCS 3.0 features, such as the CiscoWorks Home Page, depend on the UII for effective implementation. Implementing them without UII can be done, but is difficult, and should be undertaken in consultation with the CWCS team.
- Your interface should attempt to follow the User Experience guidelines, which are standard for Cisco network management applications, and useful for any application developer trying to write intuitive, learnable, and usable user interfaces. The Guidelines are also available at <http://picasso>.

CISCO CONFIDENTIAL

Adding Your Application to the CiscoWorks Home Page

The CiscoWorks Home Page (CWHP) allows users to navigate between components in a suite of applications, and serves as the launch point for both CWCS-based and third-party applications. To integrate your application with the CWHP, start by learning more about the CWHP support files and processes:

- To understand how your application will launch, see [Chapter 4, “Understanding the CWCS Execution Environment.”](#)
- To determine where your application and tasks will appear, see [Chapter 7, “Using the CiscoWorks Home Page.”](#)
- To determine what user roles will be able to access your application tasks and ensure that users authenticate before using your application, see [Chapter 10, “Using the Security System.”](#)
- To determine how you will provide online help to your users, see [Chapter 16, “Adding Online Help.”](#)

Using the Backend Services on the CWCS Server

You can use the following backend services to add additional functionality to your application:

- The Daemon Manager (also known as the Process Manager) ensures that your application handles its own processes, as well as other CWCS-based processes, efficiently. It provides the following services:
 - Initial startup following server boot or manual startup.
 - Restarting after a failure.
 - Starting only when the necessary prerequisite programs are up and running.

The Daemon Manager also includes the `cwjava` command, which provides a controlled daemon environment for CWCS-based Java server applications. For more information about the Process Manager, see [Chapter 17, “Using the Daemon Manager.”](#)

- The CWCS database engine and management tools store data used by your application, including license data, and provide backup and restore functions. For more information about data storage, see [Chapter 11, “Using the Database APIs.”](#) For more information about the backup and restore functions, see [Chapter 12, “Using Backup and Restore.”](#) For more information about licenses, see [Chapter 34, “Using the Licensing APIs.”](#)
- Event Services Software (ESS) provides an XML-based, publish/subscribe messaging service between client desktops and the CWCS Server. It replaces the Event Distribution Service (EDS). For more information about the ESS, see [Chapter 19, “Using Event Services Software.”](#) (If you are still using EDS, see [Chapter 20, “Using the Event Distribution System.”](#))
- Applications that use Job and Resource Management services can schedule an activity (a job) to occur under several conditions, including launch readiness, scheduling options, resource locks, and event notification. For more information about the Job and Resource Manager, see [Chapter 18, “Using the Job and Resource Manager.”](#)
- CWCS also supplies backend tools to add specialized capabilities to your application:
 - A generic Object Grouping Service to help you group any kind of data item, from user IDs to devices, into hierarchical picklists in your application. For more information on this service, see [Chapter 30, “Using Object Grouping Services.”](#)
 - A fully distributed interprocess communication service called CSTM. For more information on CSTM, see [Chapter 31, “Using the Common Services Transport Mechanism”](#)

CISCO CONFIDENTIAL

Using the CWCS Support Tools

Additional CWCS support services include installation tools, Java Plug-ins, and support utilities:

- The installation framework tools are free and can save your team time. Using them:
 - Allows you to consider dependencies and features such as uninstallation, thus making it easier for your package to work like other network management packages.
 - Helps you ensure that prerequisites, such as version dependencies, are set and followed.

For more information about this framework, see [Chapter 21, “Using the Installation Framework.”](#)

- To improve applet startup time and reduce the time required to download class files into the browser for each page, the Java Plug-in loads the following files locally on the client machine:
 - Java classes
 - Third-party libraries (such as JGL and Swing)
 - Images
 - Data files

For more information about the Java plug-in, see [Chapter 22, “Using the Java Plug-in.”](#)

- CWCS provides support utilities to speed the process of customer problem resolution. For more information about these utilities, see [Chapter 23, “Using the Diagnostic and Support Utilities.”](#)

Getting Up to Speed Quickly

CWCS represents a fairly extensive set of tools. Each engineer who is new to it will need time to match its capabilities against application requirements .

If CWCS is completely new to you, here is one reliable path to getting oriented quickly:

1. Start by install CD One as indicated in this chapter. Then open your browser, and get to know CWCS.
2. Read chapter 10 and get to know your security options.
3. Read chapter 32 to understand what the licensing APIs offer.
4. Skip around as needed until you feel you understand the basic CWCS design: A loosely coupled web infrastructure with built-in security, database, and administration capabilities, and some network features (such as discovery and device grouping) added in.
5. Begin planning your application, bearing in mind that it:
 - a. Should be implemented as a web application that plugs into the CWCS environment.
 - b. Must use some form of login and security.
 - c. Must use some form of licensing.
 - d. Should implement the User Interface Infrastructure, but *must* follow User Experience standards. UII is an easy way to implement UE. But in any case, CWCS is intended to support application suites, and customers expect to have a common user experience across applications.
6. Choose the other CWCS features you want to implement – database, backup and restore, the ESS event bus, JRM, etc. -- as needed. Keep in mind the service bundles you will need, because some services will run only if your application registers for the corresponding bundles.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL



PART 1

About CWCS Shared Services



CISCO CONFIDENTIAL

CHAPTER 6

Using Shared Services

Shared Services are the foundation of every application based on CiscoWorks Common Services (CWCS). They are shared among all the applications installed on a single server, and include all of the essential components of Common Services.

The following topics explain Shared Services and their supporting components:

- [Understanding Shared Services](#)
- [About the Shared Services Components](#)

Understanding Shared Services

Shared Services form the foundation of the CiscoWorks family of network management products. By “foundation”, we mean the complete set of essential services sufficient to meet basic feature requirements of all CiscoWorks network management applications, regardless of application subject area. [Table 6-1](#) list these feature requirements and the Shared Services designed to meet them.

As the foundation of the CiscoWorks product family, Shared Services are part of every CiscoWorks application (for example, Resource Manager Essentials, Campus Manager) and application bundle (for example, LMS, VMS, ITEM, SNMS). All CWCS-supported applications installed on a single server share the same set of Shared Services.

Because they are shared in this way, you cannot choose which Shared Services to install with your application. You must install all of them. Your application installation must be intelligent enough to detect whether these Shared Services are already installed on a target server before your application is installed. If Shared Services do not yet exist on the target service, are your application install must stop and prompt the user to install CWCS first. If Shared Services are installed, your application installation must not attempt to install them again.

Shared Services are included in the standard CWCS release train distribution. Unlike Per-Product Services (see [Chapter 29, “Using Per-Product Services”](#)) they are not provided as part of the “open source” (CWCS-SRC) distribution.

CISCO CONFIDENTIAL**Table 6-1 Shared Services: Feature Requirements and Services Map**

Category	Requirement	Description	Shared Service
Devices	Management	Add, import, export, delete, and synchronize devices	Device List and Credentials Repository (DCR). See the “About the Device List and Credentials Repository (DCR) Components” section on page 6-9.
	Credentials	Store and share device credentials	Device List and Credentials Repository (DCR). See the “About the Device List and Credentials Repository (DCR) Components” section on page 6-9.
	Lists	Maintain and share device lists	Device List and Credentials Repository (DCR). See the “About the Device List and Credentials Repository (DCR) Components” section on page 6-9.
Security	AAA-ACS and AAA-Non-ACS	AAA-secure interaction using ACS or non-ACS servers	MICE/CAM components in the Security System. See the “About the Security System Components” section on page 6-7.
	HTML Login	Fast, lightweight, secure user login	HTML-based user login screen (replaces the CMF applet). See the “About the Security System Components” section on page 6-7.
	User Session	Pass user session inform across Tomcat engine sessions.	MICE component from Core supports session management across different servlet engines. See the “About the Security System Components” section on page 6-7.
	Key Encryption	Encryption, key generation and agreement, Message Authentication Code (MAC)	Supported using Java Cryptography Extension (JCE). See the “About the Security System Components” section on page 6-7.
	SNMP v3 support	User key authentication per device, encrypted communications	Supports authNoPriv mode only. See the “About the Security System Components” section on page 6-7.
	Certificates	Manage and store SSL, Java and PKI certificates	See the “About the Security System Components” section on page 6-7.
	Database Encryption	Secure and encrypt database, including DB admin user name and password.	Supported using Sybase version 8 features. See the “About the Database Components” section on page 6-8.
	MSP Administration role	Enable partitioning of customer networks by device group	Supported using the MSP Admin role. See the “About the Security System Components” section on page 6-7.
Graphic User Interface	CWHP	Launch point for all applications.	CiscoWorks Home Page now provides complete GUI support, with integration down to the task level for applications that are fully UII-compliant, as explained in “About the CiscoWorks Home Page Component” section on page 6-5.

CISCO CONFIDENTIAL**Table 6-1 Shared Services: Feature Requirements and Services Map**

Category	Requirement	Description	Shared Service
Operations	Process Management	Control process startup and shutdown.	See the “About the Daemon Manager Component” section on page 6-12.
	Job Scheduling	Schedule and manage scheduled jobs and processes.	Also provides resource locking. See the “About the Job and Resource Manager (JRM) Component” section on page 6-13.
	Backup/Restore	Backup and restore data and processes.	See the “About the Backup and Restore Components” section on page 6-8.
	Application Registry	Register installed applications and configurations, check runtime dependencies	See the “About the Core Client Registry (CCR) Component” section on page 6-10.
	Message Cataloging	Log activity and errors	See the “About Diagnostic and Support Components” section on page 6-14.
	Diagnostics	Diagnose problems.	See the “About Diagnostic and Support Components” section on page 6-14.
	Connectivity Tools	Utilities for diagnosing connectivity problems (e.g., Ping, Traceroute, NS Lookup)	See the “About Diagnostic and Support Components” section on page 6-14.
	Configuration Management	Manage application configurations.	See the “About the Core Client Registry (CCR) Component” section on page 6-10.
	Packaging and Installation	Package and install applications from a single CD or set of CDs.	See the “About the Installation Framework Component” section on page 6-14.
	Integration	Allow CWCS applications to integrate with other network management applications	See the “About the Cisco Management Integration Center (CMIC) Component” section on page 6-6.
	Resource locking	Lock in-use processing resources	Provided by JRM . See the “About the Job and Resource Manager (JRM) Component” section on page 6-13
	Headroom monitoring	Monitor free system resources.	Provided by JRM and CWCS Diskwatcher. See the “About the Job and Resource Manager (JRM) Component” section on page 6-13.

CISCO CONFIDENTIAL**Table 6-1 Shared Services: Feature Requirements and Services Map**

Category	Requirement	Description	Shared Service
Other	Database	Basic application data storage.	Sybase database with SQLAnywhere. See the “ About the Database Components ” section on page 6-8.
	Database Administration	Start and stop Database engines.	Does not provide database administration. See the “ About the Database Components ” section on page 6-8.
	Messaging	Java messaging capability, communicate events and event-related notifications.	Provided on top of the Tibco event engine that is part of ESS. See the “ About the Event Services Software (ESS) Component ” section on page 6-13.
	Web Server with SSL Support	Standard web server for all applications.	Apache is the standard. See the “ About Web Server and Servlet Engine Components ” section on page 6-6.
	Servlet Engine	Supports Tomcat.	See the “ About Web Server and Servlet Engine Components ” section on page 6-6.
	Secure Shell (SSH)	Support for Secure Shell	See the “ About Web Server and Servlet Engine Components ” section on page 6-6.
	IPSec	IPSecPole support for Core-based applications.	See the “ About the Security System Components ” section on page 6-7.
	NT Services	Syslog, TFTP, RSH, and CRMLogger support for Windows	See the “ About NT Service Components ” section on page 6-15.
	Perl	Perl interpreter support	Perl version 5.00502 is supported throughout.
	Online Help	Online Help engine and files	See the “ About the Online Help Component ” section on page 6-12.

About the Shared Services Components

The following topics provide basic information for each of the Shared Services components:

- [About the CiscoWorks Home Page Component](#)
- [About Web Server and Servlet Engine Components](#)
- [About the Cisco Management Integration Center \(CMIC\) Component](#)
- [About the Security System Components](#)
- [About the Database Components](#)
- [About the Backup and Restore Components](#)
- [About the Device List and Credentials Repository \(DCR\) Components](#)
- [About the Core Client Registry \(CCR\) Component](#)
- [About the Core Logging Component](#)
- [About the Online Help Component](#)
- [About the Daemon Manager Component](#)
- [About the Job and Resource Manager \(JRM\) Component](#)
- [About the Event Services Software \(ESS\) Component](#)
- [About the Event Distribution System \(EDS\) Component](#)

CISCO CONFIDENTIAL

- [About the Installation Framework Component](#)
- [About the Java Plug-in Component](#)
- [About Diagnostic and Support Components](#)
- [About SNMP Service Components](#)
- [About NT Service Components](#)
- [About Device Center Components](#)

Each topic includes basic information about the component's purpose and features, pointers to guidelines on using it, and a list of packages on which the component has a functional dependency.

For the same information on Per-Product components, see [Chapter 29, "Using Per-Product Services"](#).

About the CiscoWorks Home Page Component

The CiscoWorks Home Page is a replacement for the CWCS Desktop that provides:

- A lightweight, high-performance Web-based GUI compatible with the Cisco UE/UII standard.
- Easy access to CiscoWorks applications via a simple, customizable "Home Page".
- Launch points for other (non-Cisco, third party, and customer-made) applications.
- MICE sharing of session data across Tomcat and other web servlet engines.
- A CMIC application registry.

For guidelines to follow when using CiscoWorks Home Page with your application, see [Chapter 7, "Using the CiscoWorks Home Page"](#). The CiscoWorks Home Page is functionally dependent on the packages shown in [Table 6-2](#).

Table 6-2 *CiscoWorks Home Page Package Dependencies*

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOhlp/pxhlp	CWCS Help (includes help engine)
CSCOhlpDM/cdone	CWCS Help Files
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjdom/jdom	JDOM XML Processing Modules
CSCOl4j/log4j	Log4j Logging Framework
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOtmt/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache Web Server, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxml4j/xml4j	IBM XML4J Parser
CSCOxrcs/xerces	Apache Xerces XML Parser

CISCO CONFIDENTIAL**Table 6-2 CiscoWorks Home Page Package Dependencies (continued)**

CSCOcore/core	Core Modules
CSTM	Common Services Transport Mechanism
UII	User Interface Infrastructure
CSCOcwHP/CWHP	CiscoWorks Home Page
CSCOcmic/CMIC	Cisco Management Integration Center

About Web Server and Servlet Engine Components

The Web Server and Servlet Engine components provide Web accessibility for all Common Services components and applications based on them. For guidelines to follow when including Web Services with your application, see [Chapter 7, “Using the CiscoWorks Home Page”](#). Web Services components are functionally dependent on the packages shown in [Table 6-3](#).

Table 6-3 Web Server and Servlet Engine Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOhelp/pxhelp	CWCS Help (includes help engine)
CSCOhelpDM/cdone	CWCS Help Files
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjaws/jaws	JSCAPE JavaAWT for widgetd
CSCOtmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Apache Web Server, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser

About the Cisco Management Integration Center (CMIC) Component

CMIC lets installed applications register their task URLs, and discover the task URLs of other applications installed on the same server. It allows these applications to discover each other and work together to provide enhanced network management integration functions they cannot provide on their own. CMIC also provides:

- An extensive search capability, which makes the CMIC Registry a lookup service for management tasks available to the user.
- A UI that allows customers to register applications manually.

For guidelines on using CMIC with your application, see [Chapter 9, “Integrating Applications with CMIC”](#). CMIC is functionally dependent on the packages shown in [Table 6-4](#).

CISCO CONFIDENTIAL**Table 6-4** *CMIC Package Dependencies*

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOhelp/pxhlp	CWCS Help (includes help engine)
CSCOhelpDM/cdone	CWCS Help Files
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOl4j/log4j	Log4j Logging Framework
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
CSTM	Common Transport Mechanism
UII	User Interface Infrastructure
CSCOwhp/CWHP	CiscoWorks Home Page
CSCOcmic/CMIC	Cisco Management Integration Center

About the Security System Components

The Security System provides secure logon and user authentication for all applications based on The Security System is non-hierarchical, session-oriented, and role-based, allowing applications to specify which of their tasks are visible to each of the user roles. For guidelines to follow when using the Security System with your application, see [Chapter 10, “Using the Security System”](#). Security System components are functionally dependent on the packages shown in [Table 6-5](#).

Table 6-5 *Security System Package Dependencies*

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOcsdb/ccsdb	Core Database
CSCOdb/db	Common Services Database (includes diskwatcher, DB wrappers)
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOhelp/pxhlp	CWCS Help (includes help engine)
CSCOhelpDM/cdone	CWCS Help Files
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjdom/jdom	JDOM XML Processing Modules
CSCOmaas/maas	maas Application Administrative Server

CISCO CONFIDENTIAL**Table 6-5 Security System Package Dependencies (continued)**

CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOtmtct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
IPSEC	Windows IPsecPol Tool (exe and dlls)

About the Database Components

The CWCS Database components provide APIs and utilities for installing, configuring and managing custom databases for your application. Among other features, the Database components allow you to set up and manipulate ODBC data sources, start and stop processes, identify versions, run scripts, and maintain backup manifests (see the [“About the Backup and Restore Components”](#) section on page 6-8). For guidelines to follow when using Database components with your application, see [Chapter 11, “Using the Database APIs”](#). Database components are functionally dependent on the packages shown in [Table 6-6](#).

Table 6-6 Database Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOdb/db	Common Services Database (includes diskwatcher, DB wrappers)
CSCOhelp/pxhelp	CWCS Help (includes help engine)
CSCOhelpDM/cdone	CWCS Help Files
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOtmtct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser

About the Backup and Restore Components

The Backup and Restore components provide a complete backup and restore function for CWCS-based applications. For guidelines to follow when including Backup and Restore with your application, see [Chapter 12, “Using Backup and Restore”](#). Backup and Restore features are functionally dependent on the packages shown in [Table 6-7](#).

CISCO CONFIDENTIAL**Table 6-7 Backup and Restore Package Dependencies**

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOcsdb/ccsdb	Core Database
CSCodb/db	Common Services Database (includes diskwatcher, DB wrappers)
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOhlp/pxhlp	CWCS Help (includes help engine)
CSCOhlpDM/cdone	CWCS Help Files
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjdom/jdom	JDOM XML Processing Modules
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOmaas/maas	maas Application Administrative Server
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOtmet/tomcat	Tomcat Servlet Engine
CSCOWeb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)

About the Device List and Credentials Repository (DCR) Components

The DCR provides a common repository for CiscoWorks-based applications to share lists of managed devices and their credentials. DCR:

- Eliminates redundant storage of this information.
- Reduces the need for application users to perform redundant maintenance operations when devices and credentials change.
- Provides a central place where users add or import new devices
- Provides for application management of this data automatically.

To use DCR with your application, see [Chapter 14, “Using the Device Credentials Repository”](#). DCR is functionally dependent on the packages shown in [Table 6-8](#).

Table 6-8 DCR Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files

CISCO CONFIDENTIAL**Table 6-8 DCR Package Dependencies (continued)**

CSCOcsdb/ccsdb	Core Database
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOhlp/pxhlp	CWCS Help (includes help engine)
CSCOhlpDM/cdone	CWCS Help Files
CSCOjpwr/jpwr	JSCAPE Power Search Classes
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOjrm/jrm	Job and Resource Manager
CSCOl4j/log4j	Log4j Logging Framework
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
CSTM	Common Services Transport Mechanism
UII	User Interface Infrastructure
CSCOcmic/CMIC	Cisco Management Integration Center

About the Core Client Registry (CCR) Component

CCR is the client registry component used by Core-based applications . It manages the installation, upgrade, patching and uninstall of Multiple Device Contoller (MDC) modules and the Core module itself. To use CCR with your application, see [Chapter 13, “Using the Core Client Registry”](#). CCR is functionally dependent on the packages shown in [Table 6-9](#).

Table 6-9 CCR Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
CSCOcsdb/ccsdb	Core Database
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjdom/jdom	JDOM XML Processing Modules
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOmaas/maas	MAAS Application Administrative Server
CSCOmd/dmgt	Daemon Manager (Process Manager)

CISCO CONFIDENTIAL**Table 6-9 CCR Package Dependencies (continued)**

CSCOperl/perl	Perl Support
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser

About the Core Logging Component

The Core Logging API allows both the Core applications (such as Multiple Device Controllers) log error, audit and application activity messages to a single file, from both C++ and Java applications. It is intended for use with the Core Client Registry (CCR), which maintains information on log accessibility and location (see the [“About the Core Client Registry \(CCR\) Component”](#) section on page 6-10). For guidelines to follow when including Core Logging with your application, see [Chapter 15, “Using the Core Logging API”](#). The Core Logging API is functionally dependent on the packages shown in [Table 6-10](#).

Table 6-10 Core Logging Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
CSCOcsdb/ccsdb	Core Database
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjdom/jdom	JDOM XML Processing Modules
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOMAAS/maas	maas Application Administrative Server
CSCOmD/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser

CISCO CONFIDENTIAL**About the Online Help Component**

The Online Help components provide access to online help for CWCS-based applications. For guidelines to follow when including Online Help components with your application, see [Chapter 16, “Adding Online Help”](#). CWCS Online Help is functionally dependent on the packages shown in [Table 6-11](#).

Table 6-11 Online Help Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOgrid/grid	Grid
CSCOhlp/pxhlp	CWCS Help (includes help engine)
CSCOhlpDM/cdone	CWCS Help Files
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOtmc/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
UII	User Interface Infrastructure
CSCOcwhp/CWHP	CiscoWorks Home Page
CSCOcmic/CMIC	Cisco Management Integration Center

About the Daemon Manager Component

The Daemon Manager (also known as the Process Manager) provides process control for applications that must:

- Monitor long-running processes.
- Restart processes that terminate abnormally.
- Start dependent processes in proper sequence.
- Start and control transient processes.

For guidelines to follow when including Daemon Manager with your application, see [Chapter 17, “Using the Daemon Manager”](#). Daemon Manager is functionally dependent on the packages shown in [Table 6-12](#).

Table 6-12 Daemon Manager Package Dependencies

Package Name	Description
CSCOmd/dmgt	Daemon Manager (Process Manager)
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail for NT)

CISCO CONFIDENTIAL**About the Job and Resource Manager (JRM) Component**

The JRM provides a general-purpose interface for scheduling application jobs and maintaining a shareable repository listing of the devices locked by particular jobs.

For guidelines to follow when including JRM with your application, see [Chapter 18, “Using the Job and Resource Manager”](#). JRM is functionally dependent on the packages shown in [Table 6-13](#).

Table 6-13 JRM Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCODB/db	Common Services Database (includes diskwatcher, DB wrappers)
CSCOeds/eds	Event Distribution System
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOmD/dmgt	Daemon Manager (Process Manager)
CSCOTmct/tomcat	Tomcat Servlet Engine
CSCOvorb/vorb	Visigenics CORBA
CSCOWeb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail)

About the Event Services Software (ESS) Component

ESS is an asynchronous, publish-and-subscribe messaging service providing distributed, loosely coupled interprocess communications. ESS is the standard CWCS service for event distribution.

The Event Distribution System (EDS; see [Chapter 20, “Using the Event Distribution System”](#)) is a predecessor of ESS. EDS is maintained in Common Services for the convenience of applications using it. *ESS and EDS are disjoint systems and do not work together. EDS has been deprecated in favor of ESS. EDS support will be withdrawn in a later version of CWCS.*

For guidelines to follow when including ESS with your application, see [Chapter 19, “Using Event Services Software”](#). ESS is functionally dependent on the package CSCOess/ess.

About the Event Distribution System (EDS) Component

EDS provides a means for sending messages from one process to another in a distributed, networked environment.

EDS is a predecessor of the standard event propagation service in CWCS: Event Services Software (ESS) (see [Chapter 19, “Using Event Services Software”](#)). *ESS and EDS are disjoint systems and do not work together. EDS is also deprecated in favor of ESS. EDS support will be withdrawn in a later version of CWCS.*

For guidelines to follow when including EDS with your application, see [Chapter 20, “Using the Event Distribution System”](#). EDS is functionally dependent on the package CSCOeds/eds.

CISCO CONFIDENTIAL

About the Installation Framework Component

The CWCS Installation Framework supplies a complete set of tools for creating full-featured installable packages, including version- and product- dependency verification, compliance with platform standards and formats, uninstallation, and patching.

For guidelines to follow when using the Installation Framework with your application, see [Chapter 21, “Using the Installation Framework”](#). The Installation Framework is functionally dependent on the packages shown in [Table 6-14](#).

Table 6-14 *Installation Framework Package Dependencies*

Package Name	Description
ITools [Windows]	CWCS Installation Framework (Windows)
ITools [SOL]	CWCS Installation Framework (Solaris)

About the Java Plug-in Component

The Java Plug-in is the Sun Microsystems product that allows Java 2 applets on CWCS web pages. This is a basic enabling technology for all CWCS applications. See [Chapter 22, “Using the Java Plug-in”](#) for specific guidelines to follow when including the Java Plug-in with an application. The Java Plug-in is functionally dependent on the package CSCOpug/plug.

About Diagnostic and Support Components

CWCS diagnostic and support utilities help customers gather data on CWCS installations and the applications installed with them. Cisco developers and customer support specialists can use this information to resolve customer problems quickly. It provides basic tools for collecting CiscoWorks Server information and packaging this information for delivery to Cisco. This edition of CWCS also includes tools for diagnosing connectivity issues and maintain log files.

For guidelines to follow when including the Diagnostic and Support tools with your application, see [Chapter 23, “Using the Diagnostic and Support Utilities”](#). The Diagnostic and Support tools are functionally dependent on the packages shown in [Table 6-15](#).

Table 6-15 *Diagnostic and Support Package Dependencies*

Package Name	Description
CSCOperl/perl	Perl Support
CSCOCORE/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail)

CISCO CONFIDENTIAL

About SNMP Service Components

CWCS SNMP Service components provide support for all basic SNMPv1, SNMPv2c, and SNMPv3 functions for both C++ and Java. For guidelines to follow when including SNMP Services components with your application, see [Chapter 24, “Using SNMP Services”](#). SNMP Services are functionally dependent on the packages shown in [Table 6-5](#).

Table 6-16 *SNMP Service Package Dependencies*

Package Name	Description
CSCOdB/db	Common Services Database (includes diskwatcher, DB wrappers)
CSCOsntp/snmp	Java SNMP APIs
SNMPv3 support	SNMPv3 Support (AuthNoPriv mode only)
C++ SNMPv3 library	These Net-SNMP libraries are being used by SNMP Set/Walk and the Management Station To Device Connectivity Tools in Device Center. Net-SNMP 5.1.1 uses OpenSSL 0.9.7d to calculate MD5/SHA-1 digests.

About NT Service Components

The NT Service components provide Windows-native support for basic functions like RSH, FTP, and Syslog. For guidelines to follow when including the NT Service components with your application, see [Chapter 25, “Using NT Services”](#). NT Services are functionally dependent on the packages shown in [Table 6-17](#).

Table 6-17 *NT Service Package Dependencies*

Package Name	Description
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail)
IPSEC	Windows IPsecPol Tool (exe and dlls)

About Device Center Components

CWCS Device Center provides a device-centric view for CiscoWorks applications. It provides device-oriented navigation, and organizes all the application tasks and reports relevant to each device around that device, making them tools launchable from a single location. Device Center can be started from the CiscoWorks Home Page or from within an application context.

For guidelines to follow when including Device Center with your application, see [Chapter 26, “Using Device Center”](#). Device Center is functionally dependent on the packages shown in [Table 6-18](#).

Table 6-18 *Device Center Package Dependencies/*

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files
CSCOcsdb/ccsdb	Core Database
CSCOess/ess	Event Services Software (includes Tibco event bus)

CISCO CONFIDENTIAL**Table 6-18** *Device Center Package Dependencies/*

Package Name	Description
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjaws/jaws	JSCAPE JavaAWT for widgetd
CSCOjre2/jre2	CWCS JRE 1.2.2
CSCOmmd/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOsnmp/snmp	Java SNMP APIs
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOcore/core	Core Modules (includes MDC, MICE, JDOM, CAM, CCR, corelogger, License files, dlls , libs, sync files, tibrv files)
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail)
IPSEC	Windows IPsecPol Tool (exe and dlls)
CSTM	Common Services Transport Mechanism
UII	User Interface Infrastructure
CSCOcwhp/CWHP	CiscoWorks Home Page
CSCOcmic/CMIC	Cisco Management Integration Center
CSCOdc/DVCR	Device Center



CISCO CONFIDENTIAL

CHAPTER 7

Using the CiscoWorks Home Page

The CiscoWorks Home Page (CWHP) is the user interface for CWCS-based network management applications. It offers an improved user interface and navigation paradigm for multiple CiscoWorks applications. It provides:

- Launch points for local and remote Cisco Works applications and their major functions.
- Easier navigation between CWHP and the applications presented on CWHP.
- Support for applications that are based on the User Interface Infrastructure (UII) and that have their own home pages.
- Support for Common Services administration.
- Launch points for external resources, including URLs on Cisco.com, custom tools and third-party applications
- Launch points for Cisco product updates.

The following topics describe CWHP and how to integrate it with your application:

- [Understanding CWHP](#)
- [Integrating Your Application with CWHP](#)

For basic information on CWHP, see the “[About the CiscoWorks Home Page Component](#)” section on [page 6-5](#).

For more information about CWHP, see:

- *CiscoWorks Home Page Functional Specification, EDCS-2818251*
- *Use Cases and Behavior of CWHP, EDCS-284828*
- *Triveni UII Integration with Core Security, EDCS-199291*
- *Mjollnir CMF 2.3 PRD, EDCS-2634301*
- *CMF 2.3 System Functional Specification, EDCS-283137*

Understanding CWHP

CWHP is the primary user interface and launch point for all local and remote CWCS administration application tasks. It is a Cisco User Interface Infrastructure (UII) application that displays banner information and tool bar items in standard locations in a web browser.

The following topics provide essential background information about CWHP and how it works:

- [About the CWHP Interface](#)

CISCO CONFIDENTIAL

- [How CWHP Works](#)
- [How CWHP Uses CMIC](#)
- [How CWHP Handles Security](#)
- [About the CWHP Runtime Structure](#)

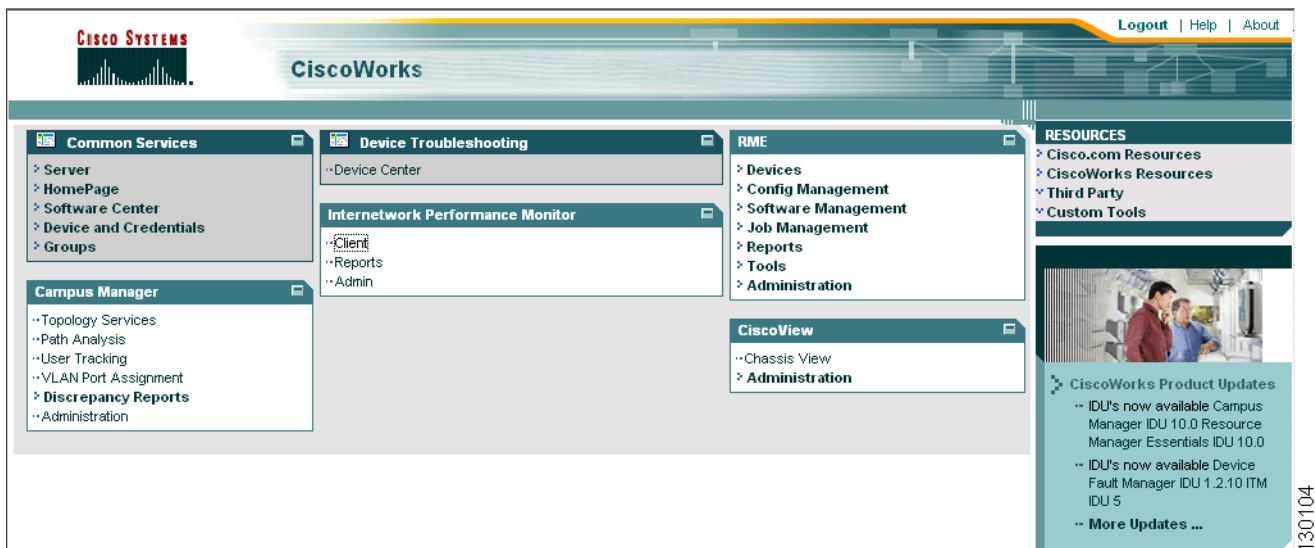
About the CWHP Interface

CWHP divides the standard UI Content Area into six main areas:

- **Common Services panel:** Provides links to all CWCS-specific functions, including server and group administration, device and credentials maintenance. This comes built-in with Common Services.
- **Device Troubleshooting panel:** Provides a launch point to the CWCS Device Center. This comes built-in with Common Services.
- **Application panels:** Display launch points for all CWCS-based applications installed on local or remote servers. The name of each application panel serves as a link to the relevant application homepage, which is launched in a new window when the user selects it.
- **LMS Setup Center panel:** Provides a launch point to the LMS Setup Center where you can configure the system settings for all the applications in one stop. This comes built-in with Common Services.
- **Resources panel:** Provides launch points for external resources (including other CiscoWorks resources, Cisco.com resources, third-party application links, and web-based custom tool links).
- **CiscoWorks Product Updates panel:** Displays informative messages about CiscoWorks product announcements, and help-related topics.
- **Tool Bar:** Allows users to logout, access online help, or display “About” information (this includes license information, version and patch level, installation date and copyright information).

Figure 7-1 shows the layout of a typical CWHP:

Figure 7-1 Typical CWHP Layout



130104

CISCO CONFIDENTIAL

CWHP uses the first three columns of the content area to display application panels. Applications get onto application panels by being registered with CMIC. The CWHP application queries the CMIC registry for all registered applications, both local and remote, at user login. It displays them based on the order they appear in the CMIC MST file.

CWHP supports applications installed on multiple servers. If the same application is found on more than one server, the instance of the application running on the local server is listed first. Instances of the application found on other servers then appear in alphabetical order by the server name.

The title of each application panel displays the application's name. The title also serves as a link to the relevant application's home page. The square expand/collapse icon shown to the right of the application's title expands/collapses the entire panel.

Application tasks are displayed as labels, in a hierarchical manner. Clicking on the label will launch the URL associated with the item in a popup window. First-level tasks for each application are listed below the title, and are shown in collapsed mode by default. Lower-level task labels are displayed only if the user manually expands the first-level item by clicking on the ">" icon. The content of each panel is limited to two levels, but there is no limit on the number of entries per level. These levels are mapped to application home-page navigation levels.

When the user clicks a task label from the hierarchy, the CWHP link to the task launches the application's home page in a new window and then selects the task by default. If an instance of the application home page is already displayed in a window, that window will be given focus and the new task will be selected.

CWHP does not perform authorization of contents displayed in application panels. It is the responsibility of applications to display proper error messages in cases where the user selects a task for which he is not authorized.

CWHP uses the last column to display information about external resources. These include CiscoWorks Resources, Cisco.com Resources, Custom Tools, Third Party Applications and Cisco Product Updates. Using the CWHP Admin UI, this section can be turned off and used for application panels.

CWHP uses the Tree Window Component to render the UI for applications. This component uses the `XbTreeWidgetStatic` JavaScript tree implementation. The `XbTreeWidgetStatic` is open-source software from Netscape and is a cross-browser JavaScript tree widget which allows flexible construction of HTML tree widgets using a simple API and HTML. DHTML manipulation of the tree is possible in browsers which support the Microsoft Internet Explorer-proprietary `HTMLDivElement.innerHTML` property. For browsers which do not support this property, `XbTreeWidgetStatic` will generate a simple hierarchical layout. `XbTreeWidgetStatic` is designed to be used before a page load event fires by writing the HTML into the document. `XbTreeWidgetStatic` is static because it cannot modify the contents of the Tree Widget after it has been written to the document. `XbTreeWidgetStatic` requires the use of JavaScript 1.2 in the implementation of its data structures.

How CWHP Works

CWHP processing involves the following steps:

1. A CWCS-based application is installed on the same server with an instance of CWCS Server. The installation script calls the CMIC Register API, passing with the call the application's MST template URN, and its host, port, and protocol. This registers the application and its home page with CMIC.
CiscoWorks applications installed on other servers, third-party applications, and custom home-grown tools are registered with CMIC by end users, using the CMIC Administration User Interface.
2. Based on the template URN, the CMIC registry fetches the application's MST file from the MST template store and updates it with the host, port, and protocol. It then replaces it in the MST template store in serialized form for easy search.

CISCO CONFIDENTIAL

3. During CWCS Server startup, CWHP loads a servlet that searches the CMIC registry for the CWHP-related integration tags shown in [Table 7-1](#). The servlet caches the CWHP-related CMIC application and task information it finds.
4. Each successful user login after that will access CWHP and invoke the CWHP servlet. The CWHP servlet, in turn, checks that it is still in sync with CMIC and, if not, recreates the CWHP cache. The servlet then fetches the application and task information from its cache.

How CWHP Uses CMIC

The CMIC registry (see [Chapter 9, “Integrating Applications with CMIC”](#)) stores data about all registered applications in the network. By querying the registry as needed, CWHP gets all the information it needs to provide launch points to these registered applications.

Applications cannot be integrated unless they have been registered with CMIC. To do this, you must create a CMIC MST template and tag your application URLs appropriately, using the task tags defined by CWHP, and register the template with CMIC. [Table 7-1](#) lists the integration tags used in the MST file to identify application tasks to CMIC.

You need not create different templates for CWHP or Device Center. All your application tasks can be part of a single template and registered with CMIC at one time.

Registration is a one-time process. You do not need to register your application again any time after initial registration. Ideally, the registration code should be part of a post-installation script executed immediately after your application is installed.

All CMIC MST templates are stored under `$NMSROOT/data/cmic/mst-templates`. You should copy your template to this location and then pass the file name during CMIC registration.

Table 7-1 CWHP Integration Tags

CWHP Function	CMIC MST Integration Tag	Use this tag to show the task in
Cisco Works Applications installed on the same server	CWHP_APP_TASK	Relevant CWHP Application panel
Cisco Works Resources	CWHP_CW_RSRC	CWHP Cisco Works Resources panel
Cisco.com Resources	CWHP_CSCO_RSRC	CWHP Cisco.com Resources panel
Cisco Works Applications installed on other servers	CWHP_OTHR_APPS	CWHP Cisco Works Other Servers panel
Third Party Applications	CWHP_THRD_PRTY	CWHP Third Party Applications panel
Custom Tools	CWHP_CSTM_TOOL	CWHP Custom Tools panel
Common Services Administration	CWHP_TASK	Common Services Panel
Device Center	DC_TOOLS	Device Troubleshooting Panel
Setup Center	systemSetup	LMS Setup Center Panel
	securitySetup	
	dataCollectionSettings	
	dataCollectionSchedule	
	dataPurge	

CISCO CONFIDENTIAL

How CWHP Handles Security

CWHP uses the UII security integration component, which integrates UII security with the CWCS security system. Using this component, CWHP can redirect security requests to the CWCS security system.

The request flow for authentication:

- User points the browser to the CWCS URL.
- The HTML login page is displayed.
- Once login is successful, redirection to the CWHP web application (running under Tomcat) happens through the help of the CMFLiaisonServlet. This servlet enables single sessions across servlet engines. If the user bookmarks his CiscoWorks Home Page and then later goes to that URL, the UII security authentication implementation will redirect the request to the HTML login page.

Authorization is required for rendering CWHP navigation menu items and this is handled by the following UII Authorization guidelines:

- Provide taskid information in the CWHP site map xml file
- Implement the isAuthorized method of UIIAuthorizeImpl to handle the authorization relevant for the CWCS security infrastructure

CWHP does not verify authorization for rendering application launch points; it assumes the user has been verified by the login. CWHP also does not perform license verification while displaying application links.

The HTML based login panel is generated through a JSP page. A JSP page renders the login panel based on the login module that is selected. SSL is used as the secure transport mechanism. SSL port will be open in non-SSL mode also, to accept login requests. The login page is served off the Tomcat servlet engine and CWCS Web Server. The SSL certificate must be generated for the CWCS Web Server at install time, as is done for CORE.

About the CWHP Runtime Structure

All CWCS shared components (such as cmic.jar) are installed under *\$NMSROOT/MDC/tomcat/lib/apps*. All shared class files are placed under *\$NMSROOT/MDC/tomcat/lib/apps/classes* directory.

All CWCS modules that provide a user interface, including CWHP, are installed in folders below the CWCS *\$NMSROOT/MDC/tomcat/webapps/cwcs* folder under Tomcat. [Table 7-2](#) shows where to find all CWHP-specific runtime files.

Table 7-2 CWHP Runtime Files

For all CWHP-related	See this folder under <i>\$NMSROOT/MDC/tomcat/webapps/cwcp</i>
JSP files	/screens/cwcp
UII action classes	/WEB-INF/classes/com/cisco/nm/cmfcwcp/ui/action
UII form bean classes	/WEB-INF/classes/com/cisco/nm/cmfcwcp/ui/form

CISCO CONFIDENTIAL

Integrating Your Application with CWHP

To integrate your application with the CiscoWorks Home Page, you must:

- Register your application and its tasks with CMIC.
- Implement security features.
- Implement any special license checks you may require
- Customize the content of the message area as needed

The following topics describe how to perform each of these tasks, as well as tips on how to proceed if you cannot fully implement UII in your application:

- [Registering Your UII-based Application with CWHP](#)
- [Implementing CWHP Security](#)
- [Implementing Special License Checks](#)
- [Handling CWHP Messages](#)
- [Migrating to CWHP](#)

Registering Your UII-based Application with CWHP

In most cases, users will use the CWHP application panels to access your application tasks. Therefore, your first task in registering your application is to create a CMIC MST file that identifies the major tasks in your application using the integration tag `CWHP_TASK`. Each task that you identify and tag in this manner will appear in the application panel for your application.

If you are using UII sub-area bar menu items with screenids, you do not have to identify every subtask on the menu in the MST. Instead, specify the sub-area bar menu item as a `TASKGROUP` and have the `TaskURL` value set to the screenid of the sub-area bar menu item. When the application is launched in the new window, the UII will take care of selecting the sub-area bar menu item and displaying the content associated with it. This also works for TOC menu items.

We recommend that you also define the custom attribute “`window_name`”, with a consistent window name as its value, within the `CWHP_TASK` integration tag. If you specify this custom attribute, CWHP will use the same window instance for all of the application’s tasks. If you do not specify this attribute, CWHP will assign unique window names for each task automatically, and each task will be displayed in a different window.

[Example 7-1](#) shows a snippet from the MST file for Campus Manager that identifies tasks to be displayed on the Cisco Works Home Page in RME window. It uses both `TASKGROUP` and `TaskURLs`.

Example 7-1 Sample MST File

```
<TASKGROUP GroupName="Devices">
  <TASKINFO TaskName="Group Management" TaskIdentity="t001" TaskDescription="Group
Management" TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/groupmanagement.do" SubmitMethod="GET" IsAPI="false">
  <INTEGRATIONTAG TagName="CWHP_TASK">
    <ATTRIBUTE Name="window_name" Value="rmewin" />
  </INTEGRATIONTAG>
</TASKINFO>
```

CISCO CONFIDENTIAL

</TASKGROUP>

Once you have completed the application MST file:

- Modify your application installation script to copy the MST file to `$NMSROOT/data/cmfcmic/mst-templates`. For information on installation scripts, see [Chapter 21, “Using the Installation Framework.”](#)
- Modify your application post-installation script to call the CMIC Register API and register this MST file with CMIC. This requires writing a java wrapper to call the Register API in a way that allows you to pass the MST file name to it. For information on the CMIC APIs, see [Chapter 9, “Integrating Applications with CMIC.”](#)

Implementing CWHP Security

Applications using CWHP can handle user authorization and authentication via the Common Services UII Security integration component, which provides a UII security implementation based on the CWCS security system. This component is packaged as `cs-uii-security.jar`, and is included in the CWCS build as an SDK tar file. It contains the following implementation classes:

- `UIIAuthenticateImpl` (`com.cisco.nm.cwcs.cwhp.uii.security`): Redirects authentication requests to CWCS security system.
- `UIIAuthorizeImpl`: Redirects authorization requests to CWCS security system.
- `CAMSystem`: A wrapper for the CWCS security system that provides UII application access to CAM security APIs.
- `securityDBLoader`: A servlet that loads UII Tasks into memory.
- `Task`: The object representation for UII Tasks.

To provide full authentication and authorization services for your CWHP-based application:

1. Place `cs-uii-security.jar` in the root of the runtime structure for your application (i.e., `$NMSROOT/MDC/Tomcat/webapps/$myapp`).
2. Specify `UIIAuthenticationImpl` and `UIIAuthorizeImpl` as your security implementation classes in your application’s `web.xml` file.
3. Define `TaskDefinition`, `RoleDefinition` and `CMFRoleDefinition` RoleMap XML files required by the CWCS security system’s CAM infrastructure. For details, see [Chapter 10, “Using the Security System.”](#)
4. Register these `TaskDefinition`, `RoleDefinition` and `CMFRoleDefinition` RoleMap XML files with CCR. For details, see [Chapter 13, “Using the Core Client Registry.”](#)
5. In the `struts-config.xml` or site map xml file, define page-level or component-level security and use the task ID string from the `TaskDefinition` XML file.

Note that:

- Your team must extend `UIIAuthenticationImpl` to support any special license requirements you may have (see).
- An additional UII `TaskDefinition` XML file is no longer needed. If you have provided a UII `TaskDefinition` file, you must change the page-level and component-level definitions in the `struts-config.xml` and `site map.xml` files, and the “taskid” attribute of `<uii:button...>` tags in JSP pages, to use the task id string from your UII `TaskDefinition` file. If not, simply specify the CAM infrastructure version instead.

CISCO CONFIDENTIAL

- Logout.jsp is no longer part of the UII security integration component. In CWCS, the Logout toolbar item is available only on the CiscoWorks Home Page. All application home pages have only a Close toolbar item.

Implementing Special License Checks

CWCS provides a complete licensing framework. This framework offers FLEXlm based licensing, which allows for a very wide range of licensing models and approaches, including feature and task-based licensing. For more information on the licensing framework, see [Chapter 34, “Using the Licensing APIs.”](#)

To integrate this smoothly with authentication, the UIIAuthenticationImpl class supports a license check method that can be performed before generating the lightweight tree navigation. If the license is not valid, the tree will not be generated.

Since the license-check is application specific, the method has been added to the UIIAuthenticationImpl class:

```
public boolean doLicenseCheck(HttpServletRequest request, HttpServletResponse response,
ServletConfig config) {
    // default implementation provided by integration component is true
    return true;
}
```

The isAuthenticated method will call doLicenseCheck after successful authentication.

Please note that it is the application team’s responsibility to extend the UIIAuthenticationImpl class and override this function based on your licensing requirements. Also be aware that, in this particular case, your team must specify this classname as the value for the init-param "UIIAuthenticateImpl" in the UIIController definition section of the web.xml file.

Handling CWHP Messages

The Product Updates panel displays product upgrades, tips, and other support information. The panel displays two types of messages:

- Standard messages: These messages are either read from the default message file or downloaded from CCO.
- Urgent messages: Urgent messages are read every 60 seconds and put at the beginning of the message queue.

Customers can add urgent messages to the Product Updates panel. Bear in mind that user messages are intended to be used by end users, not development teams. Cisco business units who want to add their marketing messages for CWCS-compatible applications or suites to the standard message file should forward their requirements to NMTG marketing.

Messages displayed in the Product Updates panel are read from files in the ***NMSROOT/lib/classpath/com/cisco/nm/cm/servlet/msgdir*** runtime directory.

The message servlet determines the type of message based on the name of the message file. [Table 7-3](#) shows the files that can be contained in the message directory.

CISCO CONFIDENTIAL**Table 7-3** Message Directory Files

File name	Description
DefaultCcoMsgFile	Used if CCO cannot be contacted to download live messages. This holds good for CWCS installations on networks that are not connected to the internet. Note To include messages about your application in the panel, work with the NMTG marketing department to get your messages added to this file.
DownloadedCcoMsgFile	Used to store messages downloaded from the CCO web site. The file is updated with a fresh download from CCO every 24 hours by default. If there is an error downloading this file, then the previously downloaded file is preserved and continues to supply messages to the panel. The DefaultCcoMsgFile is used only if DownloadedCcoMsgFile has <i>not</i> been downloaded from CCO, earlier.
.urgent	Urgent messages are read by the message servlet every 60 seconds by default and then deleted. Messages read from this file are given message-queue priority and show up in the panel immediately. There can be a delay of up to 60 seconds between the time an .urgent file is written and the time its messages appear in the message window. This is caused by the 60-second polling interval in the browser and the 30-second scanning interval for .urgent files on the server.
UserMessageFile	TBD

Migrating to CWHP

CWHP supports full integration at the task level only with web-based applications that are UII-compliant. If your application is web based (e.g., an applet, or JWS), but was not created using the User Interface Infrastructure (UII), you can only integrate with CWCS by registering your application's base URL with CWHP.

We strongly recommend that your team consider a full implementation of UII, including redesign and conversion of existing application pages, if you want to integrate with CWHP. If this is not possible, you should at least try to implement the UII navigation features. Doing so will allow you to define tasks and their navigation in the manner required to launch them directly from CWHP.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 8

Using Web Servers and Servlet Engines

Web servers and servlet engines provide basic access to all of the components that make up CiscoWorks Common Services (CWCS). They are critical CWCS components, with special capabilities, limitations, and functions.

The following topics describe how to use CWCS Web servers and servlet engines with your CWCS-enabled applications:

- [Understanding the CWCS Web Server and Servlet Engine](#)
- [Using CWCS Web Servers and Servlet Engines](#)
- [Servlet Engines and Runtime Directory](#)
- [Implications of HTML Based Login](#)

For basic information on CWCS Web servers and servlet engines, see the “[About Web Server and Servlet Engine Components](#)” section on page 6-6.

For more information about the CWCS Web Server and servlet engines, see:

- *Mjollnir - CMF 2.3 System Functional Specification: (EDCS-283137)*
- *Mjollnir - CMF 2.3 PRD (EDCS-263430) 1*

Understanding the CWCS Web Server and Servlet Engine

CiscoWorks Common Services provides a single CWCS Web Server, on which the Tomcat servlet engine runs. The CWCS Web Server uses the Apache Web Server (version 1.3.31.x or later), on both UNIX and Windows platforms, to provide the infrastructure for client/server communication.

The CWCS Web Server services HTTP and HTTPS requests from clients, and is also used to invoke CGI scripts/programs, applets, and servlets. It incorporates a customized access control module (`mod_access`) that performs session-based access control on every HTTP request. A web request must have a valid Servlet session in order to be processed, with certain exceptions allowed.

Previous versions of CWCS, such as CMF 2.2, supported components (such as the CMF Desktop and the Security system) that required the JRun servlet engine. CMF 2.2 also provided the Tomcat servlet engine for applications such as PIX MC and Kilner. All support for JRun has been dropped in this release of CWCS, and all CWCS components now run under Tomcat only.

The MICE component used for transferring session information is still available to share single session information across multiple servlet engines.

CISCO CONFIDENTIAL

The PSU component uses standard third-party products that are also used by other applications and CWCS service components. This includes the Apache Web Server, Tomcat servlet engine, and the Struts Framework. PSU has a dependency on VDS and the UII, and VDS is dependent on the CWCS Security service.

About the CWCS Web Server and SSL

The CWCS Web Server supports SSL operation, and it is enabled by default. There is an option in CWCS administration to disable or re-enable SSL mode. Note that, whenever SSL is enabled, all of the applications installed on the CWCS Server must work in SSL mode. If any applications installed on the server cannot work in SSL mode, then SSL must be disabled.

SSL is mandatory for the following applications:

- User Login
- MICE (CMFLiasonServlet)
- VMS Bundle applications

Development teams should be aware that other applications using SSL will tend to enable this mode when installed on the same server with your application. The VMS bundle, for example, automatically enables SSL for the CWCS Web Server during installation. The VMS bundle will also disable the user option to enable or disable SSL. Hence, as long as VMS is installed on a server, the server will work in SSL mode only.

Other applications can work either with or without SSL, based on configuration. In these cases, the user will have the option to enable or disable SSL. Note that a limitation of the Tomcat servlet engine and the Servlet Specification will cause any system in non-SSL mode that exposes northbound APIs (carrying user credentials) to also be in HTTP mode.

Using CWCS Web Servers and Servlet Engines

All CWCS components run under the Tomcat Servlet Engine.

About the JRE Version

This release of CWCS supports JRE versions **1.4.1_10** and **1.3.1_06**.

Components running under Tomcat are compiled under JRE 1.3.1_06. One package (CSCOjre14) has been added to allow new application code to take advantage of JRE 1.4.1_10.

JRE 1.4.1_10 is used to execute the Tomcat servlet engine.

Applications compiling for JRE 1.3.1_06 must verify compatibility with JRE 1.4.1_10. If you are compiling in 1.4.1_10, then verification is not required.

About Apache Version and Access Control

This release of CWCS supports the same version of Web Server for Tomcat, as was done in CMF 2.2. However, the versions of Apache/ModSSL/OpenSSL must be upgraded to include recent security fixes.

CISCO CONFIDENTIAL

CWCS provides custom enhancements to Apache's access control module (`mod_access`) to allow session-based access control. These are servlet sessions created when a user logs into CiscoWorks desktop. Except for certain resources, such as the login page, the modified access control module prevents service of HTTP requests that do not have an authenticated session cookie.

Ordinarily, the access-control check will throw a "Forbidden" error page if an attempt is made to access protected resources without logging in. However this was changed in CWCS based on a request from the Okena team. Instead of throwing a forbidden error, the user will be redirected to the login page, if a valid session is not present.

Servlet Engines and Runtime Directory

The runtime directory structure for CWCS components. The same structure can also be extended to CMF 3.0 based applications (e.g. RME 4.0, CM, etc).

For all Tomcat based components and applications, the runtime directory is as follows:

- All the jar files that are common across webapps are located under `$NMSROOT/MDC/tomcat/lib/apps`
- All the class files that are common across webapps are located under `$NMSROOT/MDC/tomcat/lib/apps/classes`

Each webapp defines a document base. The document base is a directory under which webapp-specific files are located. The document base is a path relative to the Tomcat home directory:

- Jar files specific for each webapp are placed under `$NMSROOT/MDC/tomcat/$document_base/WEB-INF/lib`.
- Class files specific for each webapp will be placed under `$NMSROOT/MDC/tomcat/$document_base/WEB-INF/classes`.
- JSP files using UII are located under `NMSROOT/MDC/tomcat/$document_base/WEB-INF/screens`
- JSP files not using UII are located under `$NMSROOT/MDC/tomcat/$document_base/JSP`

Runtime Structure for New Components

The fundamental runtime directory structure for this release of CWCS is the same as in CMF 2.2. All components new in this release (such as DCR, CiscoWorks Home page, Device Center, and CMIC) use the Tomcat servlet engine.

In CMF 2.2, the Tomcat root directory was `$NMSROOT/MDC/tomcat`. It would be ideal if the Tomcat root directory referred to a generic name instead of MDC. However, changing the Tomcat root directory to a new directory (such as `$NMSROOT/NG`) will involve impacts on all applications based on Tomcat, including MDCs. Therefore, it was decided to address this in a future CWCS release.

CWCS 3.0 has two new webapps based on Tomcat. One webapp includes the CiscoWorks Homepage and the user interfaces for CMIC and DCR. The other is for Device Center. All shared components are placed under `$NMSROOT/MDC/tomcat/lib/apps`. For example, `cmic.jar` is deployed under `$NMSROOT/MDC/tomcat/lib/apps/`. Similarly, any class files that need to be shared are placed under `$NMSROOT/MDC/tomcat/lib/apps/classes` directory

The new structure is explained in more detail below. The generic structure is described followed by details about individual modules. The structure is based on the Java Servlet Specification from Sun and the directories mentioned follow the Web Application structure specified by Sun. Please refer to <http://java.sun.com/products/servlet/download.html> to download the latest servlet spec.

CISCO CONFIDENTIAL**Existing Tomcat Based Components/Applications**

The runtime structure is unchanged.

**Note**

It is recommended that the existing Tomcat based components also move to the new runtime structure to have uniformity. But this decision is to be made by respective components/application teams.

New Tomcat Based Components/Applications

The runtime directory structure mentioned in this document is referenced by the TOMCAT_HOME environment variable (currently it is \$NMSROOT/MDC/tomcat). This document assumes that each application is developed and deployed as a separate webapp. All CWCS components are deployed under a single webapp.

**Note**

We do not recommended that you put any class or jar files under \$NMSROOT/lib/classpath or \$NMSROOT/www/classpath. These directories will be phased out in future releases of CWCS .

It is not recommended to put any class/jar files under \$NMSROOT/tomcat/lib as this directory is used by Tomcat servlet engine.

It is left to the component/application teams to investigate further to find whether their components will be deployed as an individual webapp or clubbed under a common webapp.

Applications deploy their files under webapps/\$app_name/ directory.

WAR Files Used by Webapps

WAR files used by webapps are located in \$NMSROOT/MDC/tomcat/webapps/

For example, CMF.war file will be deployed under \$NMSROOT/MDC/tomcat/webapps/

Jar Files Used by a Single Webapp

Jar files used by a single webapp are located under \$NMSROOT/MDC/tomcat/webapps/\$app_name/WEB-INF/lib

This could be SRC components used by a particular application. For example, RME using a particular version of UII will be deployed under \$NMSROOT/MDC/tomcat/webapps/rme /WEB-INF/lib/uii.jar

Class Files Used by a Single Webapp

Class files used by a single webapp are located under \$NMSROOT/MDC/tomcat/webapps/\$app_name/WEB-INF/classes

For example, there are some inventory APIs used by various components in RME at \$NMSROOT/MDC/tomcat/webapps/rme/WEB-INF/classes/com/cisco/nm/rme/inventory

Jar Files Shared Between Multiple Webapps

These are all shared components (CMF-R) placed under \$NMSROOT/MDC/tomcat/lib/apps

For example, cmic.jar, dcr.jar will be deployed under \$NMSROOT/MDC/tomcat/lib/apps/

CISCO CONFIDENTIAL

Class Files Shared Between Multiple Webapps

These are class files belonging to shared components placed under
\$NMSROOT/MDC/tomcat/lib/apps/classes directory

For example, LogViewer.class will be deployed under
\$NMSROOT/MDC/tomcat/lib/apps/classes/com/cisco/core/maas/server

JSP Files for Webapps

JSP files for webapps are located under \$NMSROOT/MDC/tomcat/webapps/\$app_name/screens/

For example, JSP files for DCR are located in \$NMSROOT/MDC/tomcat/webapps/cwcs/screens/dcr and for CWHP in \$NMSROOT/MDC/tomcat/webapps/cwcs/screens/cwhp and for CMIC in \$NMSROOT/MDC/tomcat/webapps/cwcs/screens/cmhc.

Action and Form Bean Classes

All Action and Form bean classes for each webapp are placed under
\$NMSROOT/MDC/tomcat/webapps/\$app_name/WEB-INF/classes/com/cisco/nm /\$app_name/ui.

Form beans are placed under /form and action classes under /action respectively.

Java Script/Images Files for Webapp

All Java script and images are placed under
\$NMSROOT/MDC/tomcat/webapps/\$app_name/js and
\$NMSROOT/MDC/tomcat/webapps/\$app_name/images

For example, for CiscoWorks Home Page, Javascript is placed under \$NMSROOT/MDC/tomcat /webapps/cwcs/js/cwhp and images under \$NMSROOT/MDC/tomcat /webapps/cwcs/images/cwhp

Other Configuration, Properties and Data Files

Other configuration, properties and data files for each webapp are placed under
\$NMSROOT/MDC/tomcat/webapps/\$app_name/etc

For example, CWHP related config files are placed in \$NMSROOT/MDC/tomcat /webapps/cwcs/etc/cwhp

Other configuration, properties and data files common across webapps are placed under
\$NMSROOT/MDC/etc/\$component_name/

For example, DCR related config files/ preferences etc placed under \$NMSROOT/MDC/etc /dcr

Runtime Structure for CiscoWorks Common Services Webapps

All CiscoWorks Common Services modules providing user interfaces are grouped under the CWCS webapp under Tomcat. These include:

- CiscoWorks Homepage UI, DCR UI, CMIC UI-related classes, JSP and JS files
- CSTM files
- UII files

JSP files related to CiscoWorks Common Services modules appear under subdirectories specific to the modules. For example:

CISCO CONFIDENTIAL

- \$NMSROOT/MDC/tomcat/webapps/cwcs/screens/dcr
- \$NMSROOT/MDC/tomcat/webapps/cwcs/screens/cwhp
- \$NMSROOT/MDC/tomcat/webapps/cwcs/screens/cmhc

All Action and Form bean classes are placed under the following directories:

- \$NMSROOT/MDC/tomcat/webapps/cwcs/WEB-INF/classes/com/cisco/nm/cmhc/cwhp /ui
/action directory will contain CiscoWorks Homepage UI related action classes
/form directory will contain CiscoWorks Homepage UI related form bean classes
- \$NMSROOT/MDC/tomcat/webapps/cwcs/WEB-INF/classes/com/cisco/nm/cmhc/dcr/ui
/action directory will contain DCR UI related action classes
/form directory will contain DCR UI related form bean classes
- \$NMSROOT/MDC/tomcat/webapps/cwcs/WEB-INF/classes/com/cisco/nm/cmhc/cmhc/ui
/action directory will contain CMHC UI related action classes
/form directory will contain CMHC UI related form bean classes

UII, CTM and OGS are all per product SRC components. So UII will be deployed under \$NMSROOT/MDC/tomcat/webapps/cwcs/WEB-INF/lib/uii.jar. Similarly CTM and OGS files (jar files) will be placed under this directory.

Runtime Structure for DCR

DCR (Device Credentials Repository) will be accessed by other applications including RME, PIX MC, IOS MC. DCR will run as a daemon. Therefore, it can be placed under \$NMSROOT/lib/server/ or \$NMSROOT/objects/server or \$NMSROOT/lib/server/dcrserver/dcrserver.jar

DCR Server jar will include OGS and CTM files.

Runtime Structure for CMHC

CMHC provides access mechanism through well-defined APIs that are part of the library files. All components or applications access CMHC using library files. The CMHC registry (database) access is taken care by library files and is transparent to the component users. Therefore, CMHC.jar will be placed under \$NMSROOT/MDC/tomcat/lib/apps for applications to use.

Runtime Structure for Device Center

Device Center will be a webapp under Tomcat: \$NMSROOT/MDC/tomcat/webapps/devicecenter/

JSP files related to Device Center will be placed under:

- \$NMSROOT/MDC/tomcat/webapps/devicecenter/screens/devicecenter

All Action and Form bean classes are placed under:

- \$NMSROOT/MDC/tomcat/webapps/devicecenter/WEB-INF/classes/com/cisco/nm/cmhc//devicecenter/ui
/action directory will contain CMHC UI related action classes
/form directory will contain CMHC UI related form bean classes

CISCO CONFIDENTIAL

Implications of HTML Based Login

The previous applet-based login panel was a bottleneck in terms of the time it took to load the login page and for the username and password fields to appear. The applet-based login used the heavyweight JAAS GUI mechanism to render the login panel. There was a lot of customer feedback asking for a reduction in the time taken for the login panel to appear.

For these reasons, the JAAS GUI mechanism was discarded and a simple HTML-based login panel, generated via a JSP page, was substituted. A JSP page is required to render the login panel based on the login module that is selected. Eliminating the JAAS GUI mechanism and the SPS transport eliminated a bulky set of classes from the login panel.

To ensure secure transport of the login credentials, CWCS must use SSL while submitting login information. The applet-based login panel used a proprietary secure-transport mechanism called SPS (Secured Packet Stream) that did not require SSL. But for HTML login, the SSL port is open in non-SSL mode also, to accept login requests. The HTML login panel requires the SSL port to be always open. This means that an SSL certificate must be generated for the CWCS Web Server at install time.

However, asking the user for certificate information twice is not acceptable, so the Core Apache install script has been changed to generate a certificate for CWCS.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 9

Integrating Applications with CMIC

Cisco Management Integration Center (CMIC) is a registry or repository for management services, enabling Cisco Management Servers to integrate with outside-world applications. CMIC acts as a service lookup for any application trying to find a particular service or set of services provided by other applications on in a network, thereby facilitating interaction amongst these application. CMIC takes care of configuring the server to integrate with applications and services residing outside the box.

CMIC Registry replaces the CMF Desktop XML Registry. It allows you to:

- Register services
- Unregister services
- Lookup services

The following topics describe how to use CMIC with your application:

- [Understanding CMIC](#)
- [Using CMIC Services](#)

For basic information on CMIC, see the “[About the Cisco Management Integration Center \(CMIC\) Component](#)” section on page 6-6.

For more information about CMIC, see:

- The *CMF Product Requirement Document CMF 3.0 PRD, EDCS-205209*
- The *CTM Functional Specification, EDCS-124878*
- The *PSU Functional Specification, EDCS-110450*
- The *CMIC Software Unit Design Specification, EDCS-137328*.

Understanding CMIC

The CMIC registry provides a Java API for registration, unregistration, and lookup of management services. These APIs are exposed via the Common Services Transport Mechanism (for details, see [Chapter 31, “Using the Common Services Transport Mechanism”](#)) so that they can be accessed over a network via North Bound (NB) APIs. All NB APIs take the extra parameter `UserCredentials`, containing user information, so accesses can be authenticated.

CMIC makes use of an XML template, called the Management Service Template (MST), to define in a coherent, standardized way each management service supplied by an application or tool. Each MST defines the attributes of the service that are relevant to other applications and services, such as the service name, description, how to invoke the service, etc. The only difference between instances of the same service is the host, port, and protocol of the instance where the service resides. The template collects all

CISCO CONFIDENTIAL

the common information, and the information that changes based on the instance is given during the CMIC service registration call. The MST file name serves as the service's Uniform Resource Name (URN); this URN is referred to in the API call for any subsequent registrations or unregistrations of the service.

The CMIC registry provides a user interface to register the following different kinds of application services:

- Third-party applications or custom tools. The UI guides the user through a series of steps that create a launch point for the application or tool from the Cisco Works Home page.
- Cisco products with pre-defined, certified MSTs. The user can select a template and specify the server on which the application is installed.
- Imported registrations from another server. For this kind of service, the UI first prompts the user for the name of the server hosting the service, along with the server's SSL port number. Once the user selects a server, the UI displays a list of applications registered with that server, and the user can then select applications that need to be registered with the local server. Imported applications having the requisite integration tags for the CiscoWorks Home page will be displayed on the local CiscoWorks Server.
- All CiscoWorks applications residing in the same server as CMIC can register during their installation by making an API call to CMIC registry. Applications not local to CW server (residing in different server) can register themselves either through the UI or using the NB bound register API exposed through CSTM.

Using CMIC Services

The following topics describe how to use CMIC services with applications:

- [Registering Applications](#)
- [Querying an Application](#)
- [Calling an Application](#)
- [Integrating CMIC with CWHP, Device Center, and Setup Center](#)
- [About the CMIC APIs](#)
- [About the Management Service Template](#)
- [Component Interaction](#)
- [About CMIC Registry Dependencies](#)
- [Sample MST File](#)

Registering Applications

Registering applications is the foremost pre-requisite to find any management service. All management services, which intend to be called by other services need to first register with CMIC.

All services willing to register need to define a Management Service Template. The template is XML based and has information about various parameters of a service like host, port, protocol, where the service resides, URL to invoke the service etc.

The users create an MST. Once the MST is created, the user is provided with a URN that identifies the MST. The name of the MST file is URN.xml

CISCO CONFIDENTIAL

This URN is used by respective services in register API along with the host, port & protocol of the host on which the service resides. CMIC registry stores all such registered services in a repository.

CMIC registry stores all such registered services in records.db repository available at *NMSROOT/objects/data/cmfcmic/registry*.

You can unregister or register applications to records.db file using command line options. This section contains the following subsections:

- [Unregistering Applications Through Command Line](#)
- [Registering Applications Through Command Line](#)

Unregistering Applications Through Command Line

To unregister an application from CMIC, enter the following command:

```
$NMSROOT\MDC\JRE\bin\java.exe -cp
$NMSROOT\MDC\tomcat\webapps\cwhp\WEB-INF\lib\cwhp.jar;
$NMSROOT\lib\classpath\cmic.jar;
$NMSROOT\MDC\tomcat\shared\lib\xerces.jar;
$NMSROOT\objects\log4j\1.1.3\log4j.jar;
$NMSROOT\MDC\tomcat\shared\lib\MICE.jar;
$NMSROOT\MDC\tomcat\shared\lib\NATIVE.jar;
$NMSROOT\lib\classpath;com.cisco.nm.cwcs.cwhp.applications.remove URN port protocol
hostname
```

where

- *NMSROOT* is the directory in which your product is installed
- *URN* is the name of the file representing the application without the .xml extension under mst-templates directory (*\$NMSROOT\objects\data\cmfcmic\mst-templates*)
- *port* is the port number used for registering the applications
- *protocol* is the protocol used for registering the application
- *hostname* is the name of host on which the service resides. If the host name is not specified, the command line utility will consider the local host name as the default value. Verify the contents of the registered templates directory (*\$NMSROOT\objects\data\cmfcmic*)

Example

To remove the Device Troubleshooting link from CiscoWorks Home page, enter the following command:

```
$NMSROOT\MDC\JRE\bin\java.exe -cp
$NMSROOT\MDC\tomcat\webapps\cwhp\WEB-INF\lib\cwhp.jar;
$NMSROOT\lib\classpath\cmic.jar;$NMSROOT\MDC\tomcat\shared\lib\xerces.jar;
$NMSROOT\objects\log4j\1.1.3\log4j.jar;
$NMSROOT\MDC\tomcat\shared\lib\MICE.jar;
$NMSROOT\MDC\tomcat\shared\lib\NATIVE.jar;
$NMSROOT\lib\classpath;com.cisco.nm.cwcs.cwhp.applications.devicecenter.1.0 1741 http afspcwsc001
```

Registering Applications Through Command Line

To register an application with CMIC, enter the following command:

```
$NMSROOT\MDC\JRE\bin\java.exe -cp
$NMSROOT\MDC\tomcat\webapps\cwhp\WEB-INF\lib\cwhp.jar;
$NMSROOT\lib\classpath\cmic.jar;
$NMSROOT\MDC\tomcat\shared\lib\xerces.jar;
```

CISCO CONFIDENTIAL

```
$NMSROOT\objects\log4j\1.1.3\log4j.jar;
$NMSROOT\MDC\tomcat\shared\lib\MICE.jar;
$NMSROOT\MDC\tomcat\shared\lib\NATIVE.jar;
$NMSROOT\lib\classpath;com.cisco.nm.cwcs.cwhp.applications.add URN port protocol hostname
```

where

- *NMSROOT* is the directory in which your product is installed
- *URN* is the name of the file representing the application without the .xml extension under mst-templates directory (*\$NMSROOT\objects\data\cmf\cmic\mst-templates*)
- *port* is the port number used for registering the applications
- *protocol* is the protocol used for registering the application
- *hostname* is the name of host on which the service resides. If the host name is not specified, the command line utility will consider the local host name as the default value. Verify the contents of the registered templates directory (*\$NMSROOT\objects\data\cmf\cmic*)

Querying an Application

A query is made to CMIC to find a management service from the registered management services. This query is based on certain search parameters

To find a particular or set of services the “query” API is used. CMIC supports query on various parameters like Application Name, Vendor name, Application Category (Fault, Configuration, Accounting, Performance, Security), Task Name etc. A query is constructed by the service requester using one or more of the parameters and sent to CMIC registry. The query can be an “AND” or “OR” on the parameters chosen for query.

Calling an Application

The query returns matched services to service requestor, the query results are processed and call to respective services are made.

CMIC registry returns a list of services matching the query to the requester. The Service requestor can process the result and launch the returned services or provide links to these services for the user to take appropriate action.

Integrating CMIC with CWHP, Device Center, and Setup Center

CMIC MST files contain all the information needed to manage and display the tasks in the Cisco Works Home Page Application Panel.

CWHP uses CMIC to register and query installed applications, Cisco Works Resources, Cisco.com resources, Cisco Works Applications installed on other servers, 3rd party applications, Custom Tools. CMIC registry has data about all registered applications in the users network, a snap shot of registry would give CWHP all the information it needs to provide launch points to various applications with the help of queries.

To integrate with CWHP, Device center, and Setup Center, you need to create a CMIC MST template and tag URLs appropriately (with integration tags) as defined by CWHP and DC, then register the template with CMIC.

CISCO CONFIDENTIAL

You need not create different templates for CWHP, Device Center and Setup Center. All tasks can be a part of single template and be registered with CMIC.

Registration is a one-time process and you do not need to register an application again after installation. The registration code can be part of post-installation script when the application is installed.

All templates must be present under *NMS-ROOT/object/data/cmfcmic/mst-templates*. Application teams can drop their template at this location and then mention the URN (same as file name with out the .xml extension) during their registration with CMIC. You can have only one MST file for an application.

The following table describes the integration tags used by the CWHP:

Table 9-1 CWHP Integration Tags

CWHP Function	CMIC MST Integration Tag	Description
Cisco Works Applications installed on the same server	CWHP_APP_TASK	Use this tag for a task in the MST file to show the task in the relevant application panel of CWHP.
Cisco Works Resources	CWHP_CW_RSRC	Use this tag of a task in the MST file to show the task in the Cisco Works Resources panel of CWHP.
Cisco.com Resources	CWHP_CSCO_RSRC	Use this tag for a task in the MST file will show this task in the Cisco.com Resources panel of CWHP.
Cisco Works Applications installed on other server	CWHP_OTHR_APPS	Use this tag for a task in the MST file will show this task in the Cisco Works Other Servers panel of CWHP.
3rd Party Applications	CWHP_THRD_PRTY	Use this tag for a task in the MST file will show this task in the Third Party Applications panel of CWHP.
Custom Tools	CWHP_CSTM_TOOL	Use this tag for a task in the MST file will show this task in the Custom Tools panel of CWHP.
Device Center	DC_TOOLS	Device Troubleshooting Panel
Setup Center	systemSetup	LMS Setup Center Panel
	securitySetup	
	dataCollectionSettings	
	dataCollectionSchedule	
	dataPurge	

Wrapper Java Code

During the application's post install, you need to call the CMIC register APIs to register the MST file with CMIC. This requires that you write a Java wrapper which will accept the MST file name. You can use the following Wrapper Java Code to register the MST file with CMIC:

```
$NMSROOT/lib/jre/bin/java -classpath
$NMSROOT/lib/classpath/cmfc.jar:$NMSROOT/MDC/tomcat/lib/apps/MICE.jar:$NMSROOT/MDC/tomcat/
lib/apps/NATIVE.jar:$NMSROOT/objects/log4j/1.1.3/log4j.jar:$NMSROOT/MDC/tomcat/lib/apps/xerces.jar com.cisco.nm.cmfc.mic.registry.CMICApplicationRegistry URN legacy action
[disable-option].
```

This code uses the arguments shown in [Table 9-2](#).

CISCO CONFIDENTIAL**Table 9-2 Register and Unregister Arguments**

Arguments	Description
URN	URN of the application to be registered.
action	Use <i>register</i> to register the application. Use this argument in the application's post install code. Use <i>unregister</i> for unregistering the application. Use this argument in the application's post remove code.
legacy	The argument indicates whether the application is a legacy application or a new application. Use 0 for new applications running under the Tomcat servlet engine. This argument helps in deciding the port and protocol for the application.
disable option	(optional) This is used to specify whether the application should be hidden in the UI when displaying the list of applications. This would help prevent the user from unregistering the application through UI. self -- Do not show application link if the application is present. But show the link when it is imported from other servers all -- Do not show the application link anywhere. none -- Default option. When unspecified, shows the link irrespective of application location

To register RME4.0 using Tomcat with default display option, use the following code in post install:

```
$NMSROOT/lib/jre/bin/java -classpath
$NMSROOT/lib/classpath/cmhc.jar:$NMSROOT/MDC/tomcat/lib/apps/MICE.jar:$NMSROOT/MDC/tomcat/
lib/apps/NATIVE.jar:$NMSROOT/objects/log4j/1.1.3/log4j.jar:$NMSROOT/MDC/tomcat/lib/apps/xerces.jar com.cisco.nm.cmf.cmhc.registry.CMHCApplicationRegistry RME4.0 0 register.
```

To register RME4.0 using Tomcat with an option to display an application link, use the following code in post install:

```
$NMSROOT/lib/jre/bin/java -classpath
$NMSROOT/lib/classpath/cmhc.jar:$NMSROOT/MDC/tomcat/lib/apps/MICE.jar:$NMSROOT/MDC/tomcat/
lib/apps/NATIVE.jar:$NMSROOT/objects/log4j/1.1.3/log4j.jar:$NMSROOT/MDC/tomcat/lib/apps/xerces.jar com.cisco.nm.cmf.cmhc.registry.CMHCApplicationRegistry RME4.0 0 register all.
```

To unregister RME4.0 using Tomcat, use the following code in post remove:

```
$NMSROOT/lib/jre/bin/java -classpath
$NMSROOT/lib/classpath/cmhc.jar:$NMSROOT/MDC/tomcat/lib/apps/MICE.jar:$NMSROOT/MDC/tomcat/
lib/apps/NATIVE.jar:$NMSROOT/objects/log4j/1.1.3/log4j.jar:$NMSROOT/MDC/tomcat/lib/apps/xerces.jar com.cisco.nm.cmf.cmhc.registry.CMHCApplicationRegistry RME4.0 0 unregister.
```

System Flow for CWHP using CMIC

- Each Cisco Works Product like RME, CM as well as Cisco Works Resources and Cisco.com resources installed on the same server where CWHP is installed, will call Register API of CMIC passing template URN, host, port, protocol.
Cisco Products Installed on other servers, 3rd Party Applications, Custom home grown tools will be created by the end user through a CMIC Administration User Interface.
- Based on the template URN, CMIC Registry will fetch the template from MST Template Store and updates the MST file with the host, port, and protocol. After registration, the MST template is stored in two places in .xml format:
 - In Registered templates store in .xml format with host, port, protocol filled in.
 - In CMIC registry in serialized format with host, port, and protocol filled in.

CISCO CONFIDENTIAL

3. During startup CWHP loads a Servlet that searches CMIC Registry on all CWHP related Integration Tags.
4. CWHP Servlet store the CWHP related CMIC information in cache to improve CWHP performance.
5. User accesses CWHP page and this invokes CWHP Servlet which in turn calls CMIC to make sure it is in sync with CMIC and if not recreate the Cache.
6. CWHP Servlet then fetches the information from Cache
7. CWHP Servlet then invokes CWHP Security UIIAuthorize Interface and returns the tasks authorized for the user. The User see the CWHP page displayed with only authorized tasks for him.

About the CMIC APIs

The following APIs are supported:

Table 9-3 CMIC APIs

API	Description
register()	To register an application with CMIC.
unregister()	To unregister an application with CMIC.
searchRegistry()	To search for a specific or list of applications in registry.
getAllRegisteredApplications()	To get a list of all registered applications.
searchRegistryReturnTree()	To look for a specific or list of applications in registry and return in a hierarchical structure.
isRegistryUpdated()	To check whether registry is updated with reference to a time stamp.

Note: Please refer the [CMIC Java doc](#) for complete list of API signatures and definitions.

About the Management Service Template

Currently Cisco runs a partner program called Cisco Management Connection, which allows third party Applications to add a launch point from CiscoWorks desktop. All such applications need to follow a certification process for their links to appear in the desktop. A similar program will be introduced to certify the MST's. This will be driven from CCO, where individual applications will be guided to create and update templates. The created template will be certified by a gatekeeper and made available to CMIC registry through PSU. All Cisco Application services will follow the same process to create and update their templates.

The information in Management Service Template is organized under various tags. The following table describes the tags and attributes used by the MST:

CISCO CONFIDENTIAL**Table 9-4 CMIC MST Tags**

Tag	Description
APPLICATIONRECORD	All attributes common to the application such as application name, application version and description
VENDORINFO	Vendor details are captured under this tag.
TASKGROUP	Used for grouping of task information.
TASKINFO	An application can have multiple tasks associated with it. Each task can be associated with one or more 'TASKGROUP' (which has a GroupName, GroupURL and GroupURLWindowName attributes) in a nested structure, or you can leave the task without associating it to a TASKGROUP.
INTEGRATIONTAG	Each task can have multiple 'INTEGRATIONTAG' elements. Applications that want to have a closer integration with other applications, define an integration tag.
ATTRIBUTES	A task can be associated with one or more integration tags. If an application task requires few custom attributes, it can make use of the 'ATTRIBUTES' element to define the custom data.
WSDL	Each task has 'WSDL' tag, the information captured here would be what input a service takes, what output it results, the format of input & output. This is defined using the Web Services Description Language (WSDL).

For more information on MST tags and attributes of each tag see, CMIC Software Design Specification, EDCS-137328.

Component Interaction

The table provides information on the components that participate in the interaction between various components.

Table 9-5 CMIC Component Interaction

Components	Description
CMIC User Interface	Handles all user related registrations and unregistration. The user can also browse through registry entries under various categories. The user interface authorizes and authenticates each user with the security services before displaying the UI.
CMIC API	Interface through which services are registered, unregistered and queried in the repository.
CSTM	All external API calls (NB API) over the network will be routed through Common Transport Mechanism (CTM). CTM is a service that accepts incoming binary requests; typically the interface will authorize all such calls and pass the request to corresponding service that handles the request (in this case CMIC). CMIC process the request API and sends the result back to CTM, which in turn sends it back to the caller. For more details on CTM please refer to CTM functional specification.

CISCO CONFIDENTIAL**Table 9-5** *CMIC Component Interaction*

Components	Description
Security Services	All authentication/authorization required by user interface and CTM interface will be fetched from security services.
PSU	Package Support Update is used to fetch MST template additions or updation to existing templates from CCO. This component is not required in the first phase.
Template Store	All downloaded MST templates from PSU will be stored here and is referred whenever there is a register/Unregister call by CMIC API.

About CMIC Registry Dependencies

The CMIC registry depends on the services shown in Table.

Table 9-6 *CMIC Dependencies*

Service	Description
Security Services	Authentication and Authorization of users, API calls received over network.
Common Services Transport Mechanism (CSTM)	Enabling API calls to be made over the network.
Desktop Services	Include UII, Web Server, and Tomcat.
Log and Trace	Logging service for recording various events and debugging.
Locking Service	For resource synchronization.
Backup and Restore	CMIC registry will provide a hook, which will do the backing up of all registered application data. The hook is called by the backup and restore service.

Sample MST File

The following file is used by RME to register launch points with CMIC:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--*****-->
<!--      Copyright (c) 2003 Cisco Systems, Inc.      -->
<!--      All rights reserved.      -->
<!--*****-->

<APPLICATIONRECORD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ws="http://schemas.xmlsoap.org/wsdl/" xsi:noNamespaceSchemaLocation="../cmic.xsd"
AppName="RME" AppVersion="4.0" TemplateVersion="1.0" AppDescription="Resource Manager
Essentials" IsCiscoCertified="false" IsCisco="false" Protocol="" Host="" Port="23"
ModifiedUser="Madan" ModifiedTime="12 " AppURL="/rme/goHome.do"
```

CISCO CONFIDENTIAL

```

AppURLWindowName="RME4.0" > <VENDORINFO VendorName="Cisco Systems" Address="170
West TasmanDr. San Jose, CA 95134 USA" Phone="(408)526-4000" Fax="(408)526-4000"
Email="tac@cisco.com" ContactURL="http://www.cisco.com"
SupportURL="http://www-tac.cisco.com"/>

<TASKGROUP GroupName="Resource Manager Essentials" GroupURL="/rme/goHome.do">

  <TASKINFO TaskName="Devices" TaskIdentity="a" TaskDescription="Device Management"
TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/deviceMgmtDflt.do" SubmitMethod="POST" IsAPI="false">

  <INTEGRATIONTAG TagName="CWHP_TASK">

</INTEGRATIONTAG>

  </TASKINFO>

  <TASKINFO TaskName="Configuration" TaskIdentity="b" TaskDescription="Configuration
Management" TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/configurationDefault.do" SubmitMethod="POST" IsAPI="false">

  <INTEGRATIONTAG TagName="CWHP_TASK">

</INTEGRATIONTAG>

  </TASKINFO>

  <TASKINFO TaskName="Image Management" TaskIdentity="c" TaskDescription="Image
Distribution" TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/swimDefault.do" SubmitMethod="POST" IsAPI="true">

  <INTEGRATIONTAG TagName="CWHP_TASK">

</INTEGRATIONTAG>

  </TASKINFO>

  <TASKINFO TaskName="Job Management" TaskIdentity="d" TaskDescription="Job Browser"
TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/JobMgmt.do" SubmitMethod="POST" IsAPI="false">

  <INTEGRATIONTAG TagName="CWHP_TASK">

</INTEGRATIONTAG>

  </TASKINFO>

  <TASKINFO TaskName="Reports" TaskIdentity="e" TaskDescription="Report Administration"
TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/CriReportJob.do" SubmitMethod="POST" IsAPI="false">

  <INTEGRATIONTAG TagName="CWHP_TASK">

</INTEGRATIONTAG>

  </TASKINFO>

  <TASKINFO TaskName="Tools" TaskIdentity="f" TaskDescription="RME Tools"
TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/toolsDefault.do" SubmitMethod="POST" IsAPI="false">

  <INTEGRATIONTAG TagName="CWHP_TASK">

</INTEGRATIONTAG>

  </TASKINFO>

```

CISCO CONFIDENTIAL

```
<TASKINFO TaskName="Admin" TaskIdentity="g" TaskDescription="Administration of RME
Server" TaskCategory="C" TaskSubCategory="C/admin" SecurityTag="nm.cm.admin"
TaskURL="/rme/adminDefault.do" SubmitMethod="POST" IsAPI="false">
  <INTEGRATIONTAG TagName="CWHP_TASK">
    </INTEGRATIONTAG>
  </TASKINFO>
</TASKGROUP>
</APPLICATIONRECORD>
```

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 10

Using the Security System

The CWCS Security system provides authentication and authorization services for CWCS-based applications. The following topics explain how to use the CWCS security system with your application:

- [Understanding CWCS Security](#)
- [Using CWCS Server Security](#)
- [Integrating a New Application](#)
- [Performing Encryption](#)
- [Stopping Eavesdropping Using SSL](#)
- [Configuring System Identity Setups](#)
- [Configuring a Cisco.com User and Password](#)

For more information about the security system or security in general, refer to the following resources:

- SSL guideline document. (ENG 123901)
- Bin:Bin Security Implementation (ENG 71441)
- Common Management Foundation (CMF 1.2) System Functional Spec. (ENG 44513)
- CMF 1.1 JEMAC Security Enhancement. (ENG 42923)
- RME Security (JEMAC) System. (ENG 28649)
- http://wwwin-eng.cisco.com/Eng/ENM/BG_10/Specs/RME_Security.doc
- Java Servlet API Specification - Version 2.1 from the Sun website. (<http://java.sun.com/products/servlet/2.1>)
- The CWCS Server end-user online help.

Understanding CWCS Security

The CWCS Server software provides some of the security controls necessary for a web-based network management system, but also relies heavily on the end user's own security measures and controls to provide a secure computing environment for CiscoWorks applications.

The CiscoWorks Server requires three levels of security to be implemented to ensure a secure environment:

- [About Client-to-Server Security](#)
- [About Server Internal Security](#)

CISCO CONFIDENTIAL

- [Using CWCS Single Sign-On](#)

About Client-to-Server Security

The Client-Server environment is architected to split an application's processing across multiple processors to gain the maximum benefit while minimizing the network traffic between machines. The key phase is to split the application processing. In Client-Server mode, each processor works independently but in cooperation with other processors .

About Server Internal Security

The shared secret system works by associating a secret character string with an alias name. If this alias name matches an existing CiscoWorks user name, then the shared secret authenticated user obtains the roles granted to the matching CiscoWorks user. If no matching CiscoWorks name exists, the authenticated shared secret user (Peer Server Account Setup) is given whatever roles are assigned to the guest user.

The secret tool shipped with CiscoWorks is a command line tool. The secretTool.pl file is located in the NMSROOT/bin directory.

Name

secretTool.pl

Description

A command line tool for managing the shared secrets used by external applications to gain authenticated access to CMF URLs.

Syntax

```
secretTool.pl [-add alias secret | -remove alias | -list | -removeSecret]
```

Input Arguments

-add alias secret	Adds a new alias to the database, or overwrite an existing one
-remove alias	Remove an existing alias from the database
-list	Lists all alias names configured in the database
-removeSecret alias secret	TBD

Examples

```
/opt/CSCOpX/bin/secretTool.pl -add myApplication mySecret  
/opt/CSCOpX/bin/secretTool.pl -remove myApplication  
/opt/CSCOpX/bin/secretTool.pl -list  
/opt/CSCOpX/bin/secretTool.pl -removeSecret
```

CISCO CONFIDENTIAL

User Name Length Restrictions

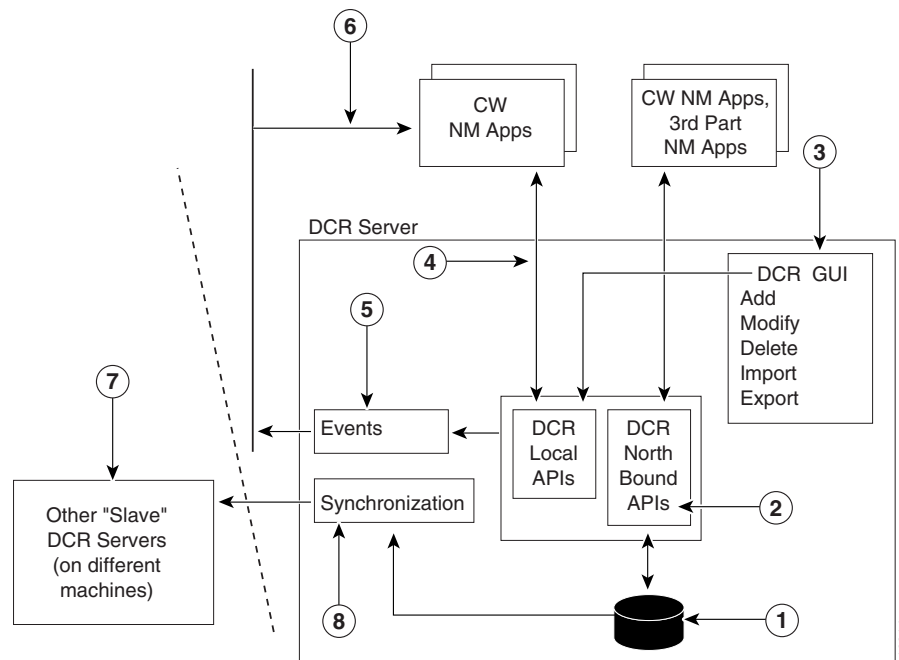
By default, the system will not allow a user to be created/authenticated when the user name length is less than 5 characters. If you need to include user names with less than 5 characters, users can set the property "validateUser=false" in NMSROOT\lib\classpath\ss.properties on Windows and NMSROOT/lib/classpath/ss.properties on Solaris. The validation for usernames lesser than 5 character will be skipped only when you set this property value to false.

Using CWCS Single Sign-On

CWCS provides a single sign-on mechanism. To use it, ensure you do the following for the initial setup:

1. Set up one of the CWCS servers as the authentication server.
2. Ensure there is trust built between the CWCS Servers using self-signed certificates. It is easier to do this with CSAMC since the certificates are rooted from the CSAMC CA. A certificate can be made trusted by adding it in the trust key store of the server. The trust key store is maintained by the certificate management framework in CWCS. It can be done using the "Add peer ciscoworks certificate" user interface.
3. Set up a shared secret with the authentication server for each CWCS Server. The System Identity Setup password is leveraged as a secret key for the single sign-on.

Figure 10-1 Login Protocol Path



The following is the path the protocol travels:

1. Visit link on a CWCS server. The server checks for a valid session.
2. The server redirects the browser to the authentication server, when there isn't a valid session for the server. The redirect URL contains a request ID and a HMAC hash using the shared secret.

CISCO CONFIDENTIAL

3. Browser redirected to the CWCS Authentication Server (AS). In case the user has already logged on, the browser has a valid session ID as cookie from AS. This session ID is sent with the redirect request.
4. The authentication server validates the request from the CWCS server by verifying the HMAC hash. An error message is thrown when validation fails. If the user has already logged on to the AS, the AS uses the session ID to fetch the user's session data, create an authentication ticket and redirect the browser back to the requesting CWCS server and the processing will continue with step 7. If there isn't a valid session, the AS sends a login page to the user.
5. The user enter the username and password and submits the login. The authentication request is sent to AS.
6. The AS authenticates the user and a user session is created. The AS generates the authentication ticket with the user data.
7. Browser is redirected to the CWCS server with authentication ticket.
8. The CWCS server verifies the HMAC in the authentication ticket to validate the request from the authentication server. On successful validation, the CWCS server generates a session context and serves the URL the user was originally navigating.

SSL is used to provide confidentiality for the transactions between the CWCS server and the authentication server. All transactions from steps 3 through 7 are protected via SSL.

How the Login Protocol Works

The login protocol works as follows:

1. The browser contacts a particular CWCS server (server1), if there are any cookies setup on the browser by server1, the browser sends them to the server1. Server1 checks the cookies to validate whether a valid session exists for the user. If a valid session does exist, then server1 sends back page requested else it continues with Login protocol.
2. Server1 generates a request ID and saves the requested URL indexed by the request ID. Server1 sends a redirect request to browser, redirecting to the central CWCS authentication server (AS). Redirect URL contains the request ID and HMAC hash based on shared secret as query parameters of the redirect URL.
3. The browser contacts AS specified in redirect URL, using HTTPS. If there is any cookies previously set by AS are stored at browser, browser sends them to server.
4. The AS checks whether there is a valid browser session based on the session ID in the cookie information, there would be a valid session in case the user has already logged in, processing continues to Step 5. In case there isn't a valid session, the AS sends back login page asking for username/password, user enters username and password and the AS verifies the username/password. On successful authentication, a valid user session context is created.
5. The AS checks validates server1 by verifying the HMAC hash using the shared secret. On successful validation, the AS generates the authentication ticket which contains the username, the request ID sent by server1 and an HMAC hash generated using the shared secret. The AS will also add server1 to the list of authenticated servers, this information would be stored in the user context.
6. AS redirects browser back to server1 with the authentication ticket added as query parameter to the end of the redirect URL.

CISCO CONFIDENTIAL

7. Browser visits the URL on server1 with the authentication ticket encoded as query parameter. Server1 validates the request from the authentication server by verifying the HMAC hash, it then generates a valid session context from the user name contained in the authentication ticket and retrieves the originally requested URL using the request ID. Server1 then proceeds to serve the original request.

How the Logout Protocol Works

The login protocol works as follows:

1. The browser initiates logout by contacting CWCS Authentication Server.
2. The AS obtains a list of servers from the user context. It generates an HTML page with Javascript containing links to logouts for each CWCS server that the user had logged into.
3. The browser executes the Javascript. The Javascript causes the browser connects to a URL for each logged-in CWCS Server that initiates CWCS logout for that server .
4. The session with the AS is also terminated.

About Server-Imposed Security

The CWCS Server provides the following security mechanisms:

- File ownership and permissions—CWCS must be installed by the system administrator and is installed as the user casuser.

On *UNIX systems* all files and directories are owned by casuser with group equal to casusers. Temporary files are created as the user casuser with permissions set to read-write for the user casuser and read only for members of group casusers. The only exception to this rule is the log files created by the CWCS web server. The CWCS web server must be started as root. Therefore, its log files are owned by the user root with group equal to *casusers*.

All backend processes are executed with a umask value of 027. This means that all files created by these programs are created with permissions equal to rwx r-x ---, with an owner and group of the user ID and group of the program that created it. Typically this will be casuser and casusers.

The *Windows* user casuser is created at installation time and given a random password to which the CWCS Daemon Manager has access. No user of the CWCS system should have to log into the Windows system as the user casuser.

Files installed by CWCS are readable by anyone logged into the Windows system, but files created in the *NMSROOT\files* folder can be read only by the casuser and administrator users.

- Executable Permissions—On *UNIX systems*, CWCS backends are executed with permissions set to the user ID of the binary file.

For example, if an executable file is owned by user “Joe”, it will be executed by the CWCS Daemon Manager under the user ID of “Joe”.

The exception is the root user ID. To prevent a potentially harmful program from being executed by the process manager with root permissions, the process manager will execute only a limited set of CWCS programs that need root privilege. This list is not documented to preclude any user from trying to impersonate these programs.

CWCS foreground processes (typically cgi-bin programs or servlets) are executed under the control of the web server’s child processes. These processes run as the user casuser.

CISCO CONFIDENTIAL

On *Windows 2000 and Windows 2003*, the runtime environment executes some backend processes as system level processes, so there are some programs that run under the system account. These programs are limited in number and controlled by entries in the Windows Registry. Only administrators have write access to these Registry entries.

- Off machine access—The CWCS process manager will not respond to requests to start, stop, register, or show status for CWCS backend processes from computers other than the CWCS Server.

About Administrator-Imposed Security

To maximize CWCS Server security, follow these system administration guidelines:

- Do not allow users who are not responsible for managing the network to have a login on the CWCS Server.
- Do not allow the CMF Server file systems to be mounted remotely with NFS or any other file-sharing protocol.
- Limit remote access (for example, FTP, RCP, RSH) to the CWCS Server to those users who are permitted to log in to the CWCS Server.
- Install CWCS on an NTFS file system because FAT file systems have no access controls.

About Server-to-Device Security

In CiscoWorks, the data is sent across in clear-text when the CiscoWorks server and devices communicate with each other. This means if someone uses a sniffer on your network, they may be able to get the device specific details (such as telnet password, and SNMP community strings) that are being passed between CiscoWorks server and a device.

CWCS implements Secure Shell (SSH) to address this vulnerability.

About Secure Shell (SSH)

SSH (Secure Shell) is a program to logon to other computers or devices over the network, to execute commands in the remote computer or device, and to move files from one computer or device to another.

It provides strong authentication and secure communications over insecure networks like internet. Currently, there are two major SSH versions available, **SSH version 1** and **SSH version 2**.

A few minor versions of SSH are also available. SSH version 1.5 is the most popular version, since it is the only version supported in Cisco IOS. Therefore, the APIs have been written specifically for SSH version 1.5. In this document, any reference to SSH means SSH version 1.5. (See ENG 167300 for details about SSH implementation in CMF 2.1.)

Using CWCS Server Security

The CWCS security system is a non-hierarchical, session-oriented, role-based system that works using registration and filtering. Each application specifies which of its tasks are visible to each of the user roles via the XML encoded application registry files.

See the [“Application Integration with CAM” section on page 10-7](#) for more details.

CISCO CONFIDENTIAL

The operating system on which the server runs is designed to enable administrators to fully protect their environment. The CWCS Server software relies on this operating system to protect its features from unauthorized use. The CWCS Server provides *some* of the security controls that are necessary for a web-based application server. However, it relies heavily on the end user's own security measures and controls to provide a secure computing environment for CWCS Server applications.

The CWCS Server provides and requires the following three levels of security:

- General security that is partially implemented by the client components of CWCS-based applications and by the system administrator.
- Server security that is partially implemented by the CWCS Server's operating system and by the system administrator.
- Application security implemented by the client and server's operating system on which CWCS-based applications reside.

About General Security

The CWCS Server provides an environment that allows the deployment of web-based network management applications. Web access provides an easy to use and easy to access computing paradigm. This is more difficult to secure than the traditional style of computing that requires a login to an operating system before applications can be executed.

The CWCS Server provides the security mechanisms (authentication and authorization) needed to prevent unauthenticated access to the CWCS Server and unauthorized access to CWCS Server applications.

Since the CWCS Server applications are capable of changing the behavior and security of your network devices, it is critical that access to the applications and servers be restricted to only those personnel who need access to applications or the data that the applications provide. To ensure a high level of security, you can:

- Limit CWCS Server logins to just the system and network administrators.
- Limit connectivity access to the CWCS Server by placing it behind a firewall. Network Services will not work unless they are behind the firewall too.

The application must integrate with CAM for client-to-server security.

Application Integration with CAM

Applications that employ the next generation features provided by CWCS integrate with the CAM infrastructure. The integration involves:

- UII Integration with CAM
- OGS Integration with CAM Device Caching.
- Application Integration with CAM.

Application Integration with CAM requires applications to do the following:

1. Applications need to register tasks in CAM.
2. Servlets must be modified to work with UII. Validation sessions are not required for servlets in the new UI/UE framework. UII also handles validation.

CISCO CONFIDENTIAL

- Components must modify the mechanism of checking authorization. There is a change in the authorization model. The new authorization model is task based as opposed to the privilege-based model in CWCS. Components must employ CAM APIs for authorization checks. The CAM APIs check authorization transparently. Components need not be aware of the underlying AAA mode (i.e. ACS or CWCS).

How CAM Cache works

CAM maintains two caches internally:

- Device cache per JVM
- User cache per user

When ACS is used, the DeviceCache object provides an interface to manipulate the local ACS cache. CAM maintains this local cache so that it does not have to go to the ACS server for every authorization request. The cache information is stored in the memory and is global to all clients.

To obtain the Device Cache created within the tomcat context:

```
DeviceCache dc = MICEKeys.getCAMObject().getDeviceCache();
    if ( dc==null || !dc.isCacheInitialized() ) {
        DeviceCache acsDevCache =
(com.cisco.core.mice.cam.DeviceCache)coreAdmin.getDeviceCache();
        acsDevCache.initDeviceCache() ;
    }
```

The cache can become stale when:

- Device or device group modifications are made in the ACS server or the MDC application
- User privilege changes on the ACS server

Sometimes, the user cache, which is not per session, may become stale when user privileges changes on the ACS Server. So the user should explicitly logout from the browser session, when the privileges changes. If the browser window is closed before the user logs out, the user cache may not be cleared and the task-to-role mapping may not be synchronized between the ACS Server and the CiscoWorks Server.

You should configure cache updates to occur when:

- Web server starts: Since the initial population of the group-to-device cache may take some time, you should perform it when the web server starts up. You should create the group-to-group cache at this time. Since this operation is common to all applications, MICE will populate the group-to-device when it starts up.
- A user logs in: When a user logs in, the application should resynchronize the group-to-device cache. This ensures that the cache does not contain any stale data at the beginning of the user's session. Since this operation is common to all MDC applications, MICE will perform this re-synchronization on behalf of the MDC applications.

**Note**

Based on requirements, you can re-synchronise the device cache created by entering

```
dc.resynchDeviceCache();
```

- The application is making authorization requests: CAM incrementally caches results of authorization requests to ACS server. When the next authorization request on the same privilege comes, the devices will be determined based on information in the cache. You can also choose to build all of the user privilege information as soon as the user logs in, through `initUserCache()`
- The user logs out: The user privilege cache should be cleared. MICE will perform this operation on behalf of the MDC application.

CISCO CONFIDENTIAL**API Level Details**

The following is a list of CsAuthServlet APIs and equivalent CAM APIs.

CsAuthServlet API Name	CWCS (CAM + CsAuthServlet)
GetAllUsers	No equivalent call in CAM.
AddUser	CsAuthServlet
RemoveUser	CsAuthServlet
ModifyUser	CsAuthServlet
AuthUser	CoreAdmin - authenticate
GetSessionID	CoreContentNexus - GetCoreID
CheckRole	CoreAdmin - authorize
GetSessionData	CoreContentNexus - GetSessionObject
GetRole	CoreAdmin - GetRole
GetCurrentUser	UserName Present in Session Object.
ValidSession	CoreContentNexus - validConnection

Authorization Checking

The following two examples explain the authorization checks performed using CsAuthServlet in previous versions of CWCS, and suggest the modifications to be made in order to integrate with CAM.

Assume that there is a component that allows a user to edit device configuration files. Typically this is an activity that a network administrator performs. The following is an example of how authorization is implemented in CWCS using CAM.

For more details and examples, see *Guidelines for CW2k Applications to Integrate with the CAM Infrastructure*, EDCS-210589.

Example 10-1 Older CWCS Model

```

{
    if ( !CsAuthServlet.checkRole(req, SA) )
    {
        out.println("ERROR: Authorization Failure");
        out.close();
        return;
    }
    Perform operation
}

```

CISCO CONFIDENTIAL**Example 10-2 Current CAM Model**

```

{
String editConfigTask = "Edit_Config";
HashMap taskList = new HashMap ();
taskList.put (editConfigTask, new Privilege (editConfigTask));
PrivilegeTask privTask = new PrivilegeTask ( "RME", taskList ,
NULL);
CoreAdmin coreAdmin = CoreAdminFactory.produce();
CoreAdmin.authorize (username, privTask, NULL);
Iterator i = privTask.getPrivileges().values().iterator();
while (i.hasNext())
{
    if (! ((Privilege) i.next()).getAuthorized())
    {
        out.println("ERROR: Authorization Failure");
        out.close();
        return;
    }
}
Perform operation
}

```

Setting Up Server Internal Security

This topic discusses the setting up server internal security and administration of the shared secret authentication mechanism built into CMF 1.2, including CiscoWorks. This mechanism allows external programs to have authenticated access to CWCS URL-based APIs. This is accomplished by executing a secure validation process based on a pre-shared secret sequence of characters. If a successful validation occurs, the servlet session associated with the connection attempt is given the security attributes needed to execute remote servlet API calls. This topic primarily covers the setup and administration of this mechanism.

Using the Shared Secret Client API

The shared secret client side API is used by external applications to gain authenticated access to CWCS server URLs. The application model is simple.

- You instantiate a SecretClient object, then pass to the secretLogin() method the host address, the secret alias name, and the secret string.
- A synchronous logon proceeds and if successful, the logon returns a session cookie that should be used in all subsequent URL calls to the host. If the login fails, some basic information is returned to determine if the problem is with the code, the network, or an actual authentication error.

The client API is provided as an unsigned jar file named SecretClient.jar which is installed with CWCS in the NMSROOT/www/classpath directory. That this jar file must be in the external program's classpath in order for the shared secret mechanism to work (copy the SecretClient.jar file to whatever machine the external program is running on if necessary).

The following code shows how to use the shared secret client API to create an authenticated session and use the returned cookie in subsequent CWCS URL calls.

CISCO CONFIDENTIAL

```
// This code will contact the SecretService running on the server, and //if everything
validates properly, it will return a valid cookie
// string. All subsequent calls to the CMF server should include this // cookie in their
requests, and this will allow server side
// authentication to operate normally.

String host = "http://xxx.xxx.x.x:xxxx";
String name = "foo";
String secret = "bar";
```

**Note**

For SSL support, replace the above three lines with the following:

```
String host = "https://xxx.xxx.x.x:xxxx";
String name = "foo";
String secret = "bar";
com.cisco.nm.cmf.ssl.initssl.initialize (true);

SecretClient sc = new SecretClient();
String cookie = sc.secretLogon( host, name, secret );

if( cookie != null )
    System.out.println( "Returned cookie=" + cookie );
else
{
    System.out.println( "Returned error code=" +
        sc.getErrCode() + ", " + sc.getErrReason() );
    return;
}

// This code will post the validSession command to the CMF CsAuthServlet as an example
// of how to call a URL using the cookie returned from secretLogon().

try
{
    String servlet = host +
        "/CSCOnm/servlet/com.cisco.nm.cmf.servlet.CsAuthServlet";
    URLConnection cnn =
        SecretClient.doPost( servlet, "cmd=validSession",
            null, cookie );
    System.out.print( "Code returned from CsAuthServlet.validSession():" );
    SecretClient.dumpResponse( cnn.getInputStream() );
}
catch( IOException e )
{
    System.out.println( "IOException " + e );
}
}
```

Client Side API Details

In your external applications, use the following shared secret methods to gain authenticated access to CMF server URLs:

- [SecretClient.secretLogon](#)
- [SecretClient.getErrCode](#)
- [SecretClient.getErrReason](#)

CISCO CONFIDENTIAL

- [SecretClient.doPost](#)
- [SecretClient.dumpResponse](#)

SecretClient.secretLogon

```
public java.lang.String secretLogon(String host, String name, String secret);
```

Client service to submit a name and secret to a remote CMF host.

Input Arguments

host	Server host to contact, such as “http://xxx.xxx.x.x:xxxx”
name	Name associated with the secret on the server
secret	Secret associated with the name on the server

Return Values

Success	Returns a validated session cookie.
Failure	Returns null. The <code>getErrCode()</code> and <code>getErrReason()</code> functions may be queried for more information.

Examples

```
/opt/CSC0px/bin/secretTool.pl -add myApplication mySecret
/opt/CSC0px/bin/secretTool.pl -remove myApplication
/opt/CSC0px/bin/secretTool.pl -list
```

SecretClient.getErrCode

```
public int getErrCode()
```

Get the last error code that occurred.

Return Values

The code returned will be one of the following integer constants:

SecretClient.OK	No error.
SecretClient.DATA_ERROR	Malformed data received due to error or possible tampering.
SecretClient.IO_ERROR	Problem communicating with remote host due to network error.
SecretClient.JRE_ERROR	Problem with JRE, expected algorithm is missing from provider.

CISCO CONFIDENTIAL

SecretClient.URL_ERROR The host server URL specified was invalid.

SecretClient.VALIDATION_ERROR Host did not validate supplied credentials.

SecretClient.getErrReason

```
public java.lang.String getErrReason()
```

Get any supplemental information available with the last error code.

Return Values

The supplemental error string, which may be null.

SecretClient.doPost

```
public static java.net.URLConnection doPost(  
    String host, String data, Object payload, String cookie )  
    throws MalformedURLException, IOException;
```

Optional helper method to execute post with handling of servlet parameters and a cookie.

Input Arguments

host Server host to contact, such as “http://xxx.xxx.x.x:xxxx.”

data Any servlet parameters to post, i.n the format: “anydata=bar&alpha=zed”, or null

payload Serializable java object to send as the payload, or null.

cookie A single cookie to send, or null.

Return Values

A URL connection object that can be used to read the response. Executes post with optional cookie and optional payload of a serialized Java object, returns connection.

Exceptions

- Malformed URL exception if the specified host server URL was invalid.
- IOException if there is a problem communicating with the remote host due to network error.

CISCO CONFIDENTIAL**SecretClient.dumpResponse**

```
public static void dumpResponse( InputStream in) throws IOException;
Optional helper method to dump a response stream of text lines.
```

Input Arguments

in InputStream to read lines from until EOS

Return Values

Dumps lines to System.out.

Setting Up Server-to-Device Security

Other applications use SSH APIs to establish secure connection between the device and CiscoWorks server. A package named CSCOSsh is provided for SSH. This package gets installed when you install CMF 2.1. The class files are available at *\$NMSROOT/lib/classpath/com/cisco/nm/cmf/ssh*

Five APIs are provided for the use of applications. SSHIO is the main class. You must initialize SSHIO before using the APIs provided for SSH.

Syntax	Description
public boolean connect	Connects to a device
public string read	Reads data from an SSH device
public void send	Executes a command in the device
public void close	Disconnects the SSH session and closes the socket
public void setDebug	Enables or disables debugging

Connecting to Device

```
public boolean connect (String HostName, int PortNumber, String UserName, String Password)
throws SshPasswdException, SshCRCEException, SshException
```

This API connects to SSH enabled device and goes into Normal User mode.

Input Arguments

HostName Name of the device or IP address.

PortNumber TCP port number to connect. Normally SSH uses port number 22.

UserName UserName to be used to connect to the device

Password Password for the user.

CISCO CONFIDENTIAL

Return Values

True	If the connection is successful and the user name/ password combination is wrong
False	Something went wrong while establishing the connection

Exceptions

SshPasswdException	Throws when the UserName/Password Combination is wrong
SshCRCEException	When Checksum of SSH packet fails
SshException	General SSH exception.

Reading From Device

```
public String read ( ) throws SshCRCEException
```

This API reads the data from SSH device. This API will be used when we want to See the Output of any device commands. This will read from the device until a device prompt appears.

Return Values

String	Returns the output of a command with device prompt
--------	--

Sending Command to a Device:

```
public void send (String Command)
```

This API executes the command in device.

Input Argument

Command	Command to be executed in device
---------	----------------------------------

Closing Connection

```
public void close( )
```

This API disconnects the SSH session and closes the socket.

Debugging

```
public void setDebug(boolean Debug)
```

This API is used to enable or disable debugging.

CISCO CONFIDENTIAL**Input Arguments**

Debug True / False

The following sample code shows how to use SSH APIs to establish a secure connection.

Example 10-3 Using SSH APIs to Establish Secure Connections

```
// This code contacts SSH enabled device and gets
// Startup Config
import com.cisco.nm.cmf.ssh.*;
class test
{
public static void main( String[] args)
{
String OP;
boolean verify= false;
SSHIO TS=null;
try
{
TS = new SSHIO();
TS.setDebug(true);
verify = TS.Connect("10.64.158.184",22,"test","test");
i
if (verify != false )
{
TS.ExecuteCommand("terminal length 0");
OP = TS.GetResult();
TS.ExecuteCommand("enable");
OP = TS.GetResult();
System.out.println(OP);
TS.ExecuteCommand("madras");
OP = TS.GetResult();
TS.ExecuteCommand("show conf");
OP = TS.GetResult();
System.out.println(OP);
TS.DisConnect();
}
else
{
System.out.println("Some thing wrong");
}
} catch (Exception e) {
System.out.println("Exception");
e.printStackTrace();
}
}
}
```

Integrating a New Application

Application integration is performed using CMIC. For details see the [“Integrating CMIC with CWHP, Device Center, and Setup Center”](#) section on page 9-4.

CISCO CONFIDENTIAL

Securing Applications

Developers should secure their applications using CAM. For details, see *Guidelines for CW2k Applications to Integrate with the CAM Infrastructure*, EDCS-210589.

The following is a deprecated mechanism of integration.

CMF provides user authentication and session tracking, but it is up to the applications to do authorization verification using the CMF security APIs.

The following topics contain guidelines for securing

- [Java Servlets](#)
- [Java Applets](#)
- [Backend Perl Script](#)
- [Java Server Pages \(JSP\)](#)

Use this set of APIs in your applications to determine if an attempt to execute an application is valid. This prevents users from by-passing the login process and attempting to execute applications directly.

If this code is not incorporated, users may be able to avoid the login/authentication process by entering the URL from the desktop.

For additional Java documentation, refer to the following location on any installed CMF Server machine: http://machine_name/javadocs/cmf/packages.html.

Securing Java Servlets

The CsAuthServlet provides the static checkRole() method for doing an authorization verification prior to performing a task. Adding a call to this method in your servlet ensures that the HTTP request is from a browser with a valid session and that the user logged in has the specified role.

**Note**

These API's are deprecated. Please use corresponding APIs in CAM.

Securing Java Applets

The CMF CsAuthServlet provides a checkRole URL for applets to do an authorization verification before starting. Calling this URL in your applet ensures that your applet is being shown in browser that has a valid user session and that the user logged in has the specified role.

Backend Perl Script

Use the following guidelines to implement a backend Perl script. The checkRole function supports both numbers and two-letter abbreviations.

CISCO CONFIDENTIAL

Description	Verifies the validity of a user's role.
Syntax	<pre>BEGIN { push(@INC, "\$ENV{'NMSROOT'}/cgi-bin/common/perl/wizard"); } use wizSecurity; ... &wizSecurity::checkRole("SA"); #Requires System Admin privileges.</pre>
Output	Verifies that a user has the appropriate privileges. If the user has privileges, the tasks display. If the user does not have the appropriate privileges to use this task, the applet displays the login panel.

**Note**

These API scripts are deprecated.

Java Server Pages (JSP)

The CMF CsAuthServlet provides the static checkRole() method for doing an authorization verification prior to performing a task. Adding a call to this method in your JSP page ensures that the HTTP request is from a browser with a valid session and that the user logged in has the specified role.

**Note**

If you are using UII, you should use UII security model to integrate with CAM. If you are not using UII, you should call CAM APIs for authorization.

Creating Auto Login Pages

If you must provide access to an application URL from outside the desktop, use the AutoLogin mechanism. A typical example of when to use this mechanism is when applications send email notifications which include a URL that can be clicked on by the recipient to view a report inside CWCS. The AutoLogin mechanism verifies that a valid session exists. If so, the specified URL is displayed. If not, the CWCS login panel is displayed and the URL is shown only after the user logs in. The URL to display must be URL encoded with the java.net. URLEncoder class or a similar function if URL parameters are included in the URLtoDisplay.

**Note**

Auto login is deprecated. Applications need to call the URL. The Apache web server will take care of forwarding to the login panel, if the session is invalid.

The Windows 2000 and Windows 2003 *casuser* account must have a random initial password that is never changed.

During an upgrade, the framework renames the old user, bin, to the new user, casuser. The user, casuser, is created during a new install by the installation framework.

The casuser does not have admin privileges on Windows 2000 or Windows 2003. It is created with User level privileges and the following additional privileges:

- SeNetworkLogonRight
- SeBatchLogonRight

CISCO CONFIDENTIAL

Performing Encryption

CWCS Security APIs can perform both symmetrical and asymmetrical encryption. The following topics discuss how to use the CWCS Java encryption APIs to encode data in your applications.

- [Handling Symmetrical Encryption](#)
- [Handling Asymmetrical \(One-Way\) Encryption](#)

Handling Symmetrical Encryption

One problem frequently encountered by application developers is the need to encrypt sensitive data (such as device credentials) when stored in a file or a database. CWCS provides the simple EncryptedObject API for doing this type of symmetrical encryption. It provides very strong encryption (Twofish/256/CBC) and the internal code is exempt from ITAR under the 15 CFR registered open source exemption. [Example 10-4](#) shows typical code using the EncryptedObject API.

**Caution**

EncryptedObject encryption is only as strong as its key management. If you hard-code the symmetric key into your code instead of obtaining it using interactive user input, a hacker could reverse-engineer your key management and extract the key. This is still far better than XOR/Base64 encoding, which is often used when tight security is not a priority.

Example 10-4 Symmetrical Encryption

```
import com.cisco.nm.cmf.security.EncryptedObject;

// To encrypt an object:

// Object to protect (NOTE: the object must be serializable!)
MyObject foo = new foo();

// Obtain the passphrase to protect access
byte[] pass = { 0, 1, 2, 3, 4, 5 ... }; // Use a decent length (32+)
// byte[] pass = passwordString.getBytes(); // Another way...

// Encrypt and encapsulate it in serializable EncryptedObject instance
EncryptedObject e = new EncryptedObject( foo, pass );

// Encrypted object e is now safe, do whatever with it including read/write to/from disk
...

// To unencrypted the object later:

// Obtain the passphrase to protect access
byte[] pass = { 0, 1, 2, 3, 4, 5 ... }; // Use a decent length (32+)
// byte[] pass = passwordString.getBytes(); // Another way...

// Obtain the encrypted object
EncryptedObject e = ...; // read from disk, e.g.

// Recover the encapsulated encrypted object
MyObject foo = (MyObject)e.getObject( pass );
```

CISCO CONFIDENTIAL

Handling Asymmetrical (One-Way) Encryption

Use asymmetrical encryption when you do not need to recover encrypted data. A common use for this type of encryption is for UNIX password files. Asymmetrical encryption is performed using a message digest algorithm which is a secure one-way hashing function. To perform one-way encryption on a string, the `java.security.MessageDigest` class can be used (available in Java 1.1). [Example 10-4](#) shows a typical implementation of this kind.

Example 10-5 Asymmetrical Encryption

```
import java.security.MessageDigest;

String myString = new String("Data to encrypt");
Byte[] encryptedData;

// instantiate and initialize MessageDigest object
try {
    MessageDigest md = MessageDigest.getInstance( "SHA" );
}
catch( NoSuchAlgorithmException ) {
    ...
}
md.reset();

// set data to encrypt
md.update( myString );

// get encrypted data
encryptedData = md.digest();
```

Stopping Eavesdropping Using SSL

Using a network sniffer is a common practice among malicious hackers. Hackers look for decipherable data transferring across the network that can be used to hack into unsuspecting networks or provide them with confidential information. Data between a customer's browser and CiscoWorks is in sent across the network in clear-text. This means if someone uses a sniffer on your network they may be able to see any data that is being passed between CiscoWorks and the user. This is detrimental due to the amount of data in CiscoWorks that would be very useful to a malicious hacker (topology information, device passwords, and so on).

Why Use SSL in CWCS?

One way customers can protect themselves from eavesdropping is through SSL (Secure Socket Layer). SSL encrypts the transmission channel between the client and server.

Using SSL does come at a slight cost. The encryption technology used in SSL is math intensive, meaning it can slow down the computer's processor. Also, since the transmission needs to be encrypted/decrypted, it can add latency to the connection.

CISCO CONFIDENTIAL

SSL Support in CWCS

CWCS provides secure access between the client browser and management server and also between the management server and devices. It uses SSL encryption to provide secure access between the client browser and management server, and Secure Shell (SSH) to provide secure access between the management server and devices.

Users can enable or disable SSL from the CWCS desktop, depending on their need to use secure access between the client browser and the management server. CWCS has provisions to manage security certificates, both self-signed certificates and certificates issued by third-party certificate agencies (CA). To learn more about enabling and disabling SSL, and the certificate management functions, see the *User Guide for CiscoWorks CommonServices*.

What Kind of SSL Support is Available in CWCS?

All applications in CWCS support SSL. If the web server is SSL enabled, you can invoke the applications through Hypertext Transfer Protocol Secure (HTTPS).

The web server supports SSL version 2, 3, and TLS version 1. We support strong 128-bit encryption to enable customers to have maximum security. Since the export laws have recently been relaxed, the strong 128-bit encryption does not affect the exportability of CWCS.



Note

If you have non-SSL compliant applications installed on the server, SSL cannot be enabled in CWCS.

SSL-Enabling Your Application

Though the the CWCS Web Server had the ability to support SSL connections (HTTPS) over port 1742 in addition to the default HTTP (over port 1741) from CMF version 1.2, the CiscoWorks code did not work over this connection. However, there are provisions to make the CiscoWorks code work correctly over the SSL session.

You can find the details for enabling your application to work over SSL in the document *CW2000 applications over SSL Sessions*, EDCS ENG-123901.

Configuring System Identity Setups

Applications use System Identity Setup to authenticate processes on remote CiscoWorks Servers.

Suppose there are two CiscoWorks servers A and B. You configure x, x1 and x2 as Peer Server Account Setup on A and y, y1, and y2 as Peer Server Account Setup on server B. To set up a System Identity Setup, you have to configure one Peer Server Account Setup in A, say x, as the System Identity Setup in B.

The System Identity Setup should have the necessary privileges to perform the desired tasks. Also, the System Identity Setup should be configured with the same password.

In ACS mode, the System Identity Setup needs to be configured with the required privileges in ACS.



Note

During installation, CWCS prompts for setting up a System Identity Account username and password.

CISCO CONFIDENTIAL

The following is an example of CommonTrustUser API.

```
public String getCommonTrustUser () : Returns the system identity setup.

public void setCommonTrustUser (String username, String secret) :Sets the system identity
setup and writes into the file, Input : UserName and SecretKey

public char[] getCommonTrustUserKey () :Returns system identity setup key if it is
configured.

public String getCommonTrustUserAndKey () : Returns system identity setup and key in the
format username:key
```

Configuring a Cisco.com User and Password

Certain CWCS features require access to the cisco.com web site work. For example, CiscoWorks must be configured with a cisco.com account to use when downloading new and updated packages. This user account is also used with so-called CCO APIs, which allow your CWCS application to access cisco.com. [Example 10-6](#) shows some typical uses of CCO APIs

Example 10-6 CCO APIs

```
public String getCCOLogin() : Returns the CCO Login

public void setCCOUser (String username, String secret): Sets the cco user and writes into
the file Input : UserName and SecretKey

public String getCCOPassword (): Returns cco password if it is configured.

public String getCCOInfo (): Returns cco login and password in the format login:password
```



CISCO CONFIDENTIAL

CHAPTER 11

Using the Database APIs

CWCS database APIs are used primarily for installing and configuring custom databases. These APIs hide the configuration, platform details, and database management processes from applications. They allow applications to:

- Manipulate ODBC data sources
- Start and stop database processes and identify database versions
- Run SQL scripts
- Manipulate backup manifests

The following topics describe how to create a custom database and how to use the CWCS database APIs in your applications:

- [Understanding the CWCS Database, page 11-2](#)
- [Setting Up a New Database, page 11-6](#)
- [Performing a Quick Integration, page 11-17](#)
- [Using the Sybase Database, page 11-18](#)
- [Debugging and Troubleshooting the Database, page 11-33](#)
- [Database API Command Reference, page 11-37](#)

For more information about the CWCS database APIs, refer to:

- *CMF 1.2 Database Functional Specification*, EDCS ENG-54964
- *Database Security*, EDCS ENG-80264

For details on backing up and restoring the database, see [Chapter 12, “Using Backup and Restore.”](#)

The Sybase SQL Anywhere Studio 9.0.0 Core Documentation Set is available online at <http://sybooks.sybase.com/onlinebooks/group-sas/awg0900e>. Recommended titles in this set include:

- *Adaptive Server Anywhere Database Administration Guide*
- *Adaptive Server Anywhere Getting Started*
- *Adaptive Server Anywhere Programming Guide*
- *Adaptive Server Anywhere SQL Reference Manual*
- *Adaptive Server Anywhere SQL User's Guide*
- *Introducing SQL Anywhere Studio*
- *MobiLink Synchronization User's Guide*
- *Ultralite Database User's Guide*

CISCO CONFIDENTIAL

Understanding the CWCS Database

The following topics describe various aspects of the CWCS database:

- [What's New in This Release](#)
- [Understanding the Tools](#)
- [Database Access Methods](#)
- [Understanding the NMTG Database Delivery Process](#)

What's New in This Release

The following changes have been implemented in this release:

- **JConnect Upgrade:** Sybase declared JConnect 4.2 End-of-Life in December 2002, and no longer supports it. Accordingly, CWCS 3.0 drops support for JConnect 4.2, and upgrades support for JConnect 5.2 to JConnect 5.5.
- **Sybase Upgrade:** The Sybase database engine on Solaris and Windows was upgraded to 9.0.0 + EBF. The EBF versions as of release were 9.0.0.1364 (Solaris) and 9.0.0.1366 (Windows).

Understanding the Tools

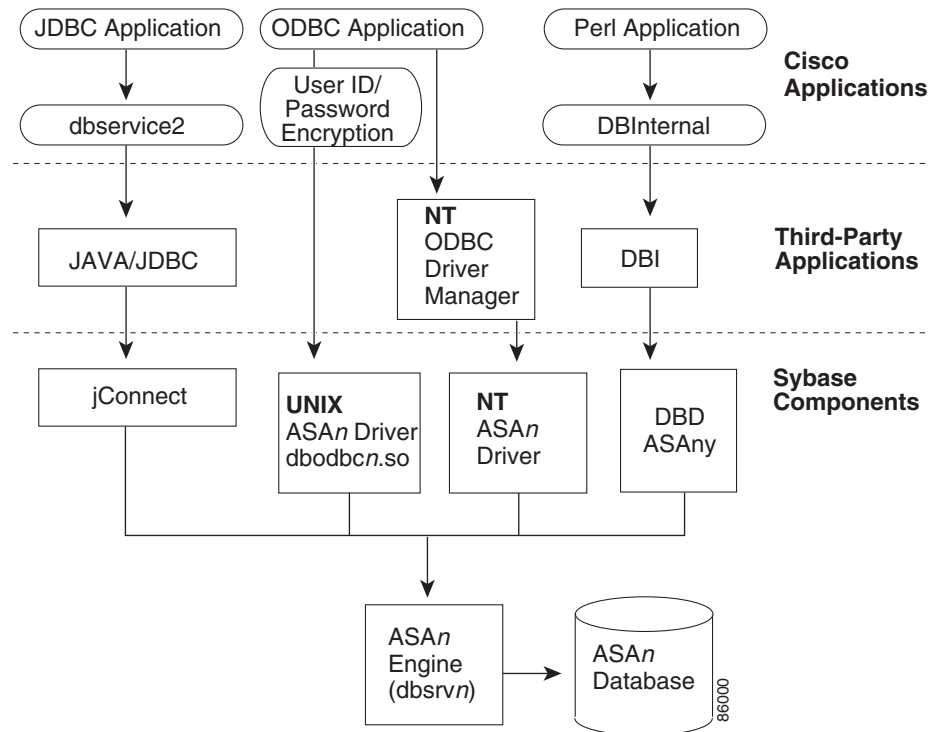
These third-party tools are required to implement the CWCS database:

Tool	Version	Description
jConnect	5.5	JDBC commands use this Sybase component to access the database engine.
JDBC	1.2	Java Database Connectivity—A set of Java APIs that provide universal data access for the Java programming language.
ODBC	2.x ODBC, 3.510 of ODBC Driver Manager	Open Database Connectivity—A standard call level interface developed by Microsoft and based on the SQL AccessGroup CLI specification. ODBC is a database-independent C language API that consists of a driver manager (supplied by the operating system) and drivers for each database vendor.
SQL Database Engine	V9.0.0 + EBF	Adaptive Server Anywhere 9.0.0 is the current version of the Sybase database present in CWCS. EBF versions as of release were 9.0.0.1364 (Solaris) and 9.0.0.1366 (Windows). This database supports the ODBC API on UNIX and Windows platforms. Includes access from Java via JDBC, C/C++ via ODBC, and Perl via DBI.
DBInternal	CWCS component	DBInternal is the link that Perl uses to access the database (on Windows platforms only).

CISCO CONFIDENTIAL**Database Access Methods**

CiscoWorks applications can communicate with the CWCS database using these methods: ODBC, JDBC, and Perl. [Figure 11-1](#) shows how each method accesses the CWCS database.

Figure 11-1 Database Access Applications



The following topics describe these access methods:

- [Types of Database Servers](#)
- [JDBC Access Methods](#)
- [ODBC Access Methods](#)
- [Perl Access Methods](#)
- [Connection Strings](#)

Types of Database Servers

There are two types of database servers:

- **dbsrv9**—The network version. This allows unlimited concurrent connections, provides multi-user use, and supports client/server communication across a network.
- **dbeng9**—The personal version. This allows up to 10 concurrent connections. Use this simpler version when:
 - The CWCS Server is down, and therefore no other connection is on.
 - A simple task such as restore or backup needs to connect to the database to check the data.

CISCO CONFIDENTIAL

JDBC Access Methods

JDBC (Java Database Connectivity) is a set of Java APIs that provide universal data access for the Java programming language. JDBC provides a standard interface between your application and the database server.

The JDBC commands use the Sybase component, jConnect, to access the database engine. When an application makes a database connection, the classes in dbservice2 retrieve the connection information from the DbServer.properties file, including the database user ID and password, to construct the JDBC URL.

The dbservice2 Java classes also provide commonly-used JDBC methods. Use these Java methods, which sit on top of the JDBC APIs, *instead* of the JDBC API to shield any database-related changes from high-level applications.

Related Topics

See:

- The “About the Database Property Files and Settings” section on page 11-12.
- Sun’s Java training site at the following URL:
<http://developer.java.sun.com/developer/onlineTraining/Database/JDBCShortCourse/index.html>

ODBC Access Methods

ODBC (Open Database Connectivity) is a standard call level interface (CLI) developed by Microsoft and based on the SQL AccessGroup CLI specification. An industry standard, ODBC is a database-independent C language API that consists of a driver manager (supplied by the operating system) and drivers for each database vendor.

The Sybase Adaptive Server Anywhere database engine supports the ODBC API on UNIX and Windows platforms (see [Figure 11-1](#)). To access ODBC functions, the applications must be compiled and linked with the appropriate import library file. The Sybase ODBC driver makes the connection using the database user ID and password stored in the .odbc.ini file on UNIX platforms or the system registry on Windows platforms.

Related Topics

See Microsoft’s online SDK site at the following URL:

<http://msdn.microsoft.com/downloads/sdks/platform/database.asp>

Perl Access Methods

DBI is the Perl ODBC interface. It defines a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used. DBI does not access any particular database; instead, it locates and loads the applicable driver modules.

The database user ID and password are stored in the .odbc.ini file on UNIX platforms or the system registry on Windows platforms.

Perl applications use the following drivers to access the database:

- On Windows platforms, Perl applications use the DBInternal driver (see [Figure 11-1](#)) to connect to the DBI module.

CISCO CONFIDENTIAL

- On Solaris platforms, Perl applications use the DBD driver. The DBD (Database Driver) modules contain the vendor libraries and can access the actual databases; there is one DBD module for every database. For example, the driver for the Adaptive Server Anywhere database is DBD::ASAny.

Related Topics

See the Comprehensive Perl Archive Network (CPAN) web site at the following URL:
<http://www.perl.com/CPAN-local/README.html>

Connection Strings

Applications need to establish a connection to the database before they can interact with the database. A connection requires, at a minimum, the user ID, password, and database name. For SqlAnywhere databases and ODBC programs, this information is specified as a single parameter in the form of a connection string:

- SqlAnywhere connection strings—Used for any registered or unregistered database. SqlAnywhere databases use the form `ENG=xx;CWEUID=xx;CWEPWD=xx`, where:
 - ENG is the database engine name
 - CWEUID is the encrypted database user ID
 - CWEPWD is the encrypted database password

This connection string can be made more specific by adding the DSN parameter to refer to a database attached to the database engine.



Note If the encryption flag is ON, applications must use the encrypted user ID and password keywords, CWEUID and CWEPWD. If, however, the application still uses the plain text user ID and password (the encryption flag is OFF), the old connection string format that uses UID and PWD will still work.

- ODBC connection strings—Used only if the database is registered. ODBC connection strings use the form `DSN=xx;CWEUID=xx;CWEPWD=xx`, where the DSN parameter refers to a data source that contains a detailed definition of the data source. This includes the name of the database engine, engine start up parameters, the path to the database root file, and so on. The data source can also store the user ID and password. In this case, the connection string can be just `DSN=xx`. The data source information is kept in the registry on Windows platforms, and in the `.odbc.ini` file on UNIX systems.

Understanding the NMTG Database Delivery Process

Table 11-1 provides a summary of the database files that require special handling during the delivery process. *If you are not using ClearCase and the NMTG installation processes, you will need to create a similar process for delivering these files.*

CISCO CONFIDENTIAL**Table 11-1** NMTG Database Delivery Phases and Files

Delivery Phase	Special Files
Create these files and place them in the ClearCase vob.	<p>These files will be copied to the CD by the build process:</p> <ul style="list-style-type: none"> • <i>\$NMSROOT/databases/orig/odbc.tmplorig</i>: Contains the factory password for the initial database (<i>db_name.dborig</i>), the user ID and password, the engine name, and the port ID. The <i>odbc.tmplorig</i> file must be duplicated to the <i>odbc.tmpl</i> file during installation. • <i>\$NMSROOT/databases/orig/db.dborig</i>: Contains the default database for your module. To create a database, use the <i>dbinit</i> command. <p>Refer to later sections for details on using the <i>dbinit</i> command and changing the default username/password for this database.</p>
During installation	<p>The CWCS database is registered and System Services are enabled by default. The registration and installation process:</p> <ul style="list-style-type: none"> • On Solaris only: Renames the <i>.odbc.tmpl</i> file to <i>.odbc.ini</i> and populates it with the user ID, password, and other database information for each database engine. • Renames the database template files (<i>db.dborig</i>) and copies them to <i>\$NMSROOT/databases/dsn.db</i>. For example, <i>\$NMSROOT/databases/cmfc/orig/cmfc.dborig</i> is renamed and copied to <i>\$NMSROOT/databases/cmfc/cmfc.db</i>. • On Solaris systems: Populates the <i>dmgtd.conf</i> file with the database engine command for each database. It also adds the database monitor command for each database. • Updates the <i>DBServer.properties</i> file with the URLs for the installed suites as well as the database credential information. • On Windows only: Updates the Windows registry (if applicable) with the ODBC service information.
After installation	<p>The following files have been modified:</p> <ul style="list-style-type: none"> • Windows registry—For Windows platforms. Contains the ODBC services for the database engine. • <i>.odbc.ini</i>—For Solaris systems. Contains the ODBC services for the database engine. • <i>dmgtd.conf</i>—A Daemon Manager file that contains the database engine and database monitor commands. • <i>DBServer.properties</i>—Contains the JDBC URL for the installed database.

Setting Up a New Database

The following topics describe the files, settings, and processes required to create a new database:

- [Creating the ODBC Database Definition File](#)
- [Creating the Backup Manifest Files](#)
- [About the Database Property Files and Settings](#)
- [Managing the Database Engine](#)

If you want to set up a new database quickly, see the “[Performing a Quick Integration](#)” section on [page 11-17](#).

CISCO CONFIDENTIAL

Creating the ODBC Database Definition File

The ODBC DSN uses the database definition file, `.odbc.ini`, to provide database-specific information to ODBC applications. This file contains the user ID, password, and other database information.

- On Windows platforms, the ODBC definition is located under the Windows Registry. For example, the registry key for CWCS is:

```
HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\cmf
```

The key `CWEUID` contains the encrypted user ID, and the key `CWEPWD` contains the encrypted password.

- On Solaris platforms, the ODBC definition is located under `$NMROOT/.odbc.ini`. The environment variable `ODBCINI=$NMROOT/.odbc.ini` is defined by the Daemon Manager.

The following topics describe the procedures for creating a database template file:

- [Creating the Database Template File](#)
- [Creating the `odbc.tmplorig` Template File](#)
- [Enabling Database Password Encryption](#)

Creating the Database Template File

The registration and configuration process uses the database template file, `.odbc.tmplorig`, to create the `odbc.tmp` and `.odbc.ini` files on Solaris, and the registry entries on Windows:

- On Solaris platforms, it renames the `.odbc.tmp` file to `.odbc.ini` and populates it with the user ID, password, and other database information for each database engine.
- On Windows platforms, the `.odbc.ini` file does not exist. Instead, the `odbc.tmp` file is used to populate the Windows registry with the user ID, password, and other database information for each database engine.



Note If you are a developer working inside NMTG, follow the procedure in the [“Creating the `odbc.tmplorig` Template File”](#) section on page 11-7.

Creating the `odbc.tmplorig` Template File

If you are a developer working inside NMTG, create an `odbc.tmplorig` database template file. The automated build processes use this file to create the corresponding `.odbc.ini` file on the target system. Use the following procedure to create the `odbc.tmplorig` file.

Step 1 Create the `odbc.tmplorig` file using these conventions:

- **File Name:** `odbc.tmplorig`
- **ClearCase Location:** `/vob/enm_cmf/share/databases/cmf/`

Step 2 Include the following lines (on Solaris platforms, expand the file names to their full path):

```
UID=cmfDBA
PWD=c2kY2k
Start=dbsrv9
DatabaseName=cmfDb
EngineName=cmfEng
```

CISCO CONFIDENTIAL

```

CommLinks=tcPIP{HOST=localhost;DOBROADCAST=NO;ServerPort=43441}
CWENCRYPTION=YES
AutoStop=yes
# note __ values are not passed through for odbc registration
# These __ values are skipped by odbcdsn.pl.
# These __ values are used for configuring db engine startup parms.
__Cache=8
__DbNTSvcLongName=CiscoWorks Cmf database engine
JdbcDriver=com.sybase.jdbc2.jdbc.SybDriver
DmPrefix=Cmf

```

The JdbcDriver line populates the JdbcDriver entry in DBServer.properties. For more information about this entry, see the [“Creating the Database Template File”](#) section on page 11-7. To register with custom switches, or specify a JDBC driver, see the [“Customizing the odbc.tmpl File”](#) section on page 11-8.

Step 3 Make these changes:

- Line 1: Replace *cmfDBA* with your user ID.
- Line 2: Replace *c2kY2k* with your password.
- Line 3: Replace *dbsrv9* with the database engine name.
- Line 4: Replace *cmfDB* with your database name. Do not include the absolute path; the full path is constructed and inserted into the .odbc.ini file. For more information about this process, see the [“Understanding the NMTG Database Delivery Process”](#) section on page 11-5).
- Line 5: Replace *cmfEng* with your engine name.
- The CommLinks line contains database engine command line parameters. Replace “43441” with the port ID for your database. To determine which port ID number to use, see the [“Managing the Database Engine”](#) section on page 11-13.
- If you do not want to enable database password encryption, remove the line, CWENCRYPTION=YES (see the [“Enabling Database Password Encryption”](#) section on page 11-9).
- Adjust other lines as needed:
 - The Cache line is a database engine command line parameter that defines the size of the cache (default = 8 M).
 - The __DbNTSvcLongName defines the Windows Service name to be the CiscoWorks database engine.
 - The JdbcDriver line populates the JdbcDriver entry in DBServer.properties.
 - The Dmprefix is the prefix registered for this database in the CWCS Daemon Manager.

Customizing the odbc.tmpl File

If you want to register your database with custom flags, you need to include the `-Switches` entry in `odbc.tmpl`, with the additional switches you want specified on the same line as the property name.

If the `_Switches` property is not present, the database will be registered with the following default switches on:

- On Windows platforms: `-m -ti 0 -gm 100`
- On Solaris platforms: `-q -m -ti 0 -gm 100`

CISCO CONFIDENTIAL

Note that if you specify custom switches and also want the default switches, you must include both on the `_Switches` line.

The following will always be present and need not be included on the `_Switches` line:

- `-c` for cache size (this is taken from the `_Cache` line) .
- `-n` for the engine name and database file name
- On Solaris only: The option `-s $ENV{PX_FACILITY}` is always added. You need not enter it with your custom switches.

For example:

```
CWEUID=r0wicBlFHWg=
CWEPWD=vJa9p8EtilQ=
Start=dbsrv9
DatabaseName=cmfDb
EngineName=cmfEng
CommLinks=tcip{HOST=localhost;DOBROADCAST=NO;ServerPort=43441}
AutoStop=yes
CWENCRYPTION=YES
# note __ values are not passed through for odbc registration
# These __ values are skipped by odbcdsn.pl.
# These __ values are used for configuring db engine startup parms.
__Cache=8
__Switches= -u -p
__DbNTSvcLongName=CiscoWorks Cmf database engine
DmPrefix=Cmf
```

You also have the option to specify any compatible JDBC driver. To do so, add the following entries in the `odbc.tmpl` file:

```
"JdbcDriver=myDriver"
"DataSourceUrl=MySource"
```

When the database is registered using `configureDb.pl`, these entries will be added to `DBServer.properties`. If these entries are absent, `Jconnect` will be used by default.

Enabling Database Password Encryption

The encrypted password and username are stored in the `.odbc.ini` file on Solaris and the registry entry on Windows. It is also stored in the `odbc.tmpl` and `DBServer.properties` files.

To enable password encryption, add the `CWENCRYPTION` flag to the `odbc.tmpl` file. For example:

```
UID=cmfDBA
PWD=c2kY2k
Start=dbsrv9
DatabaseName=cmfDb
EngineName=cmfEng
CWENCRYPTION=YES
CommLinks=tcip{HOST=localhost;DOBROADCAST=NO;ServerPort=43441}
AutoStop=yes
__Cache=8
__DbNTSvcLongName=CWCS Cmf database engine
```

The `CWENCRYPTION` flag:

- Indicates that an application wants to encrypt its identification information.
- Has two possible values, `YES` or `NO`. The default is `NO`.

We strongly recommend that all applications call the following command during installation:

CISCO CONFIDENTIAL

```
$NMSROOT/objects/db/conf/ChangeDbPasswd.pl dsnname newpwd
```

Put this command in the following location:

- On Solaris platforms: `postinstall`
- On Windows platforms: the `rul` file

You can call `ChangeDbPasswd.pl` before or after the database is installed.

`ChangeDbPasswd.pl` validates passwords. Valid passwords must:

- Have a minimum of five and a maximum of 128 characters.
- Use alphanumeric characters (a-z, A-Z, 0-9) only. No special characters (e.g., #, \$, %) or spaces are allowed.
- Not have a number as the first character.

If you are setting a new database password during the application install, we recommend that you validate the submitted password. If your install submits an invalid password, `ChangeDbPasswd.pl` will generate an appropriate information message and will *not* change the password.

Related Topics

See the:

- [“Creating the Database Template File” section on page 11-7.](#)
- [“Creating the odbc.tmplorig Template File” section on page 11-7.](#)

Creating the Backup Manifest Files

Backup manifest files are ASCII text files used by the CWCS backup and restore framework. These files contain a list of the database files or directories to be backed up.

There are two types of backup manifest files:

- The *database* backup manifest file contains a list of database file names. The `backup.pl` script uses this list to determine which database files to back up. The database backup manifest file is stored in the directory `$NMSROOT/backup/manifest/suite/database/orig/dsn.txt`, where:
 - `$NMSROOT` is the directory in which the product will be installed.
 - `suite` is the name of your application or suite. Often, this is the same as the `dsn`. For example: For CWCS, the `suite` and `dsn` are both “`cmf`”, but for Campus Manager, the `suite` is “`campus`” and the `dsn` is “`ani`”.
 - `dsn` is the the data source (database) name.
- The *application* backup manifest file contains a list of directories and files where application-specific data is stored. The `backup.pl` script uses this list to determine the application data to back up. The application backup manifest file is stored in the directory `$NMSROOT/backup/manifest/suite/app/orig/datafiles.txt`, where:
 - `$NMSROOT` is the directory in which the product will be installed.
 - `suite` is the name of your application or suite, as for the database backup manifest file.
 - `app` is the name of the application or module within the suite. These usually vary. For example: For Resource Manager Essentials, the `suite` is “`rme`”, but the `app` may be “`configArchive`”.

The following topics explain how to create both types of backup manifest files:

- [Creating the Database Backup Manifest File](#)

CISCO CONFIDENTIAL

- [Creating the Application Backup Manifest File](#)

**Note**

Use of the CWCS backup and restore framework requires that Campus Manager, ACLM, and DFM developers change their application backup manifest directory structure. For details, see the [“CWCS Backup” section on page 12-1](#).

Related Topics

See the:

- [“Enabling the CWCS Database Engine” section on page 11-37](#).
- [“Using CWCS Backup” section on page 12-1](#).
- [“Running CWCS Backups” section on page 12-3](#).
- [“backup.pl” section on page 12-10](#).

Creating the Database Backup Manifest File

Use the following procedure to create the backup manifest files for your database:

-
- Step 1** Create a *dsn.txt* file, where *dsn* is the name of your database.
- Step 2** Include the following lines in *dsn.txt*, replacing all occurrences of “cmf” with your database name:
- ```
[cmf]
root=${ENV{NMSROOT}}/databases/cmf/cmf.db
```
- Step 3** Copy the *dsn.txt* file to the following directory in the runtime tree:  
`$NMSROOT/backup/manifest/suite/database/orig/dsn.txt`
- Where:
- `$NMSROOT` is the directory in which the product was installed.
  - *suite* is the name of your application or suite. This is sometimes the same as the *dsn*. For example: For CWCS, the *suite* and *dsn* are both “cmf”, but for Campus Manager, the *suite* is “campus” and the *dsn* is “ani”. When you install CWCS, the cmf.db database is loaded and enabled by default.
  - *dsn* is the name of your database.

The final runtime location of this file will be `$NMSROOT/backup/manifest/suite/database/` (that is, the `/database` subdirectory that is the parent of `/orig`).

---

**Creating the Application Backup Manifest File**

Use the following procedure to create the backup manifest file for your application’s files:

- 
- Step 1** Create a *datafiles.txt* file.
- Step 2** On separate lines, list the application paths and files to be backed up. For example:
- ```
${ENV{NMSROOT}}/lib/classpath/com/cisco/nm/cmf/servlet/cwpass
${ENV{NMSROOT}}/lib/classpath/sso.properties
${ENV{NMSROOT}}/lib/classpath/com/cisco/nm/dcr/dcr.ini
```

CISCO CONFIDENTIAL

```
$ENV{NMSROOT}/lib/eds/filter/namedfilter.str
```

The `$ENV(NMSROOT)` value must precede each line (the `configureDb.pl action=install` call will replace it with the actual installation directory).

Step 3 Copy `datafiles.txt` to the following directory in the runtime tree:

```
$NMSROOT/backup/manifest/suite/app/orig/datafiles.txt
```

Where:

- `$NMSROOT` is the directory in which the product was installed.
- `suite` is the name of the suite containing your application (often, this is the same as the `dsn`).
- `app` is the name of your application. Often, this is the same as `suite`, especially for standalone applications.

The final runtime location of this file will be `$NMSROOT/backup/manifest/suite/app/` (that is, the `/app` subdirectory that is the parent of `/orig`).

About the Database Property Files and Settings

The following topics describe the two types of database property files:

- [About the Database Server Property File](#)
- [About Private Property Files](#)

About the Database Server Property File

`DBServer.properties` is the database server properties file. It contains the configuration parameters for Java Database Connectivity (JDBC) application database functions such as various debug levels, timeouts, and sleep periods, the port the database service module listens to for socket-based requests, the maximum number of database connections, and so on.

File Name	<code>DBServer.properties</code>
Runtime Location	<code><i>\$NMSROOT</i>/www/classpath/com/cisco/nm/cmfdbservice</code> (where <code><i>\$NMSROOT</i></code> is the directory in which the product was installed).
ClearCase Location	<code>enm_cmf/share/classes/client/com/cisco/nm/cmfdbservice/orig</code>
Example	To see an example of the <code>DBServer.properties</code> file, refer to the <code>CodeSamples</code> directory on the CWCS SDK CD.

Typically, you should *not* change the contents of the `DBServer.properties` file; the information in this file is created by `CMFEnable.pl`. For each registered database, the `CMFEnable` script adds several lines to this file. For example, these lines were appended for the CMF database:

```
### dbconnection for cmf ###
DBConnection.userName.cmf=cmfDBA
DBConnection.password.cmf=c2kY2k
DBConnection.dataSourceUrl.cmf=jdbc:sybase:Tds:localhost:43441?SERVICENAME=c
mfDb
DBConnection.jdbcDriver.cmf=com.sybase.jdbc2.jdbc.SybDriver
```

You might, however, need to modify this file to change:

CISCO CONFIDENTIAL

- The JDBC connection information
- The jConnect tuning parameters
- Any debug level, sleep, or timeout values

To make changes to the DBServer.properties file:

-
- Step 1** Use an ASCII editor such as Notepad to update the file.
- Step 2** Stop the database engine process.
- Step 3** Restart the database engine process.
-

Related Topics

- [“Enabling the CWCS Database Engine” section on page 11-37.](#)
- [“Starting a Database Engine” section on page 11-25.](#)
- [“Stopping a Database Engine” section on page 11-27.](#)

About Private Property Files

Some applications include the database password in a private, application-specific property file. For CWCS to recognize this password, these applications must adhere to the following conventions:

- Add the location of the private property file to the `odbc.tmplorig` file. The key name is `PropertyFile`. The value must include the path of its private property file relative to `$NMSROOT`, the directory in which the product was installed. For example, the `odbc.tmplorig` file for ANI includes this line:

```
PropertyFile=etc/cwsi/ANIServer.properties
```

- The key name for the database password in its property file must be `“DB.password”`. For example, the `ANIServer.properties` file includes this line:

```
DB.password=cwsiPWD
```

Related Topics

- [“Creating the ODBC Database Definition File” section on page 11-7.](#)

Managing the Database Engine

The following topics describe some important database engine management tasks:

- [Understanding Port IDs](#)
- [Creating a Database Port](#)
- [Changing the Database Port](#)
- [Dynamically Allocating a Port ID](#)

CISCO CONFIDENTIAL

Understanding Port IDs

Because CWCS uses a database that supports cross-network operations, every database engine must have its own port ID. Every network server must define a unique port ID as a TCP/IP parameter.

At install time, the Installer framework makes a call to the CWCS service bundles enabling mechanism, CMFEnable.pl. This Perl script uses the values in the CommLinks line in the database template file, .odbc.ini, to assign each database engine its own port ID. For more information about the database template file, see the [“Creating the ODBC Database Definition File” section on page 11-7](#).

All newly-developed databases must define a CommLinks value. The format of the CommLinks definition is:

```
CommLinks=tcPIP{HOST=localhost;DBBROADCAST=NO;ServerPort=portid}
```

As part of the database registration process, CMFEnable.pl reads the .odbc.ini file and populates the port ID from the CommLinks entry to several places:

- **Database Server Command Line**

A network server is started with a TCP/IP protocol. The command line is stored in the following locations in the runtime tree:

- On Solaris Platforms, it is stored in:
`$NMSROOT/conf/dmgt/dmgt.d.conf`
- On Windows platforms, it is stored under this Windows system entry:
`HKEY_LOCAL_MACHINE \SYSTEM\CurrentControlSet\Services`

- **The ODBC connection parameters**

The ODBC driver detects a database RPC protocol and port ID from entry CommLinks. This entry is defined:

- On Solaris Platforms, in this file:
`$NMSROOT/.odbc.ini`
- On Windows platforms, in the Windows system registry entry with this key:
`HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\dsn\CommLinks`

- **The JDBC URLs**

For both Windows and UNIX platforms, the JDBC application reads the database URL definition from this file:

```
$NMSROOT/www/classpath/com/cisco/nm/cmfdbservice/DBServer.properties
```

The CWCS installation code checks to see if the port ID is available, and gives a warning if it is not. This warning, however, does not stop the installation process.

The following topics describe how CWCS allocates port IDs:

- [Creating a Database Port](#)
- [Changing the Database Port](#)
- [Dynamically Allocating a Port ID](#)

Related Topics

See the [“Enabling the CWCS Database Engine” section on page 11-37](#).

CISCO CONFIDENTIAL**Creating a Database Port**

CWCS reserves ports 43441 through 43549 for database servers. [Table 11-2](#) shows the ports permanently allocated to CWCS and existing applications. To permanently allocate a database port for your application, contact the CWCS database team via the support alias, embu-db-interest@cisco.com.

If you are using Solaris platforms and CWCS-supplied scripts like `configureDB.pl`, the scripts will check for and select a free port automatically, as follows:

1. If you specified a port in `odbc.tmpl`, the scripts will check `/etc/services` to see if this port is already in use. Then:
 - a. If there is no entry for the port in `/etc/services`, the scripts will assume the port is free and will select it.
 - b. If there is an entry for the port, the script will:
 - Assume that another application is using it.
 - Pick a port from the dynamic range 43461-43480.
 - Check to see if that port is free. It will select the first port in that range for which there is no entry in `/etc/services`.
 - If none of the ports in the range 43461-43480 are free, the script returns an error.
2. If there is no port specified in `odbc.tmpl`, a free port from the range 43461-43480 is picked up and selected.
3. The scripts enter the selected port in `/etc/services`. For example:

```
cscocmfdb          43441/tcp          # CSCO NM cmf database
```

Note that port checking and `/etc/services` updates are *not* available if you are using Windows platforms or non-CWCS configuration scripts. For details, consult DDTs defects CSCsa09950 and CSCsa11233.

Table 11-2 *Permanently Allocated Database Server Ports¹*

Port	Application	Contact
10033 ²	IDS MC	Patti Abkowitz (abkowitz)
10033 ²	Sec Mon	Patti Abkowitz (abkowitz)
10033 ²	PIX MC	Joel Klein (jfklein)
10033 ²	AUS	Jared Smith (jarsmith)
10033 ²	QPM	Oren Fridler (ofridler)
10033 ²	Router MC	Yardena Meymann (ymeymann)
43441	CWCS	Vikram Rao (vikram)
43442	RME	Vivi Zhang (vzhang)
43443	Campus (ANI)	Suresh Pathamadai (pssuresh)
43444	Service Level Manager	Sajan Mathew (samathew)
43445	Kilner (FH)	Pavan Kumar Mirla (pavankm)
43446	Kilner (Inventory)	Pavan Kumar Mirla (pavankm)
43447	Kilner (EPM)	Pavan Kumar Mirla (pavankm)
43448	Kilner (AMA)	Shiva Shankar (shaj)

CISCO CONFIDENTIAL**Table 11-2** Permanently Allocated Database Server Ports¹ (continued)

Port	Application	Contact
43449	PIF	Shiva Shankar (shaj)
43450 ³	PIF HTTP Server	Shiva Shankar (shaj)
43451	WAN Performance Utility	Mathangi Kuppusamy (mathangi)
43453	Performance Monitor	Wei W. Wang (weiwa)
43454	Security Auditor	Mark Lu (malu)
43455	RME NG	Venunadh Veerala (vveerala)
43458	CVM	Auro Dharmapuram (auro)
43459	Pairpoint	Subbu Chandrasekaran (csubrama)
44341	IPM	Pavan Kumar Mirla (pavankm)

1. For the most current list, see http://www.in-embu.cisco.com/embueng/database_ports.htm.
2. These databases share the same database server (SqlCoreDBServer).
3. Not a database, but this port is permanently allocated to the application specified.

Changing the Database Port

To use another port number for your database engine, you can use the `configureDb` utility from the command line to assign a new port ID. Use this utility only when:

- The Daemon Manager is down.
- The database is registered and enabled.



Caution

Use this utility with caution. It is not intended to provide another interface for database registering, and should be called only after the subsystem is registered. For more information about the `configureDb` utility, see the “[configureDb.pl](#)” section on page 11-53.

To change the port number:

Step 1 At the command line, enter (all on one line):

```
/bin/perl $NMSROOT/objects/db/conf/configureDb.pl action=upgrade dsn=$dsn portid=$portid
```

Where:

- `$NMSROOT` is the directory in which the product was installed.
- `$dsn` is your database name.
- `$portid` is the new port ID you want to assign.

The utility updates all required files and Windows system registry entries.

CISCO CONFIDENTIAL**Dynamically Allocating a Port ID**

Database port IDs can be assigned dynamically during installation. Dynamic port ID assignments, however, can increase development and troubleshooting times dramatically.

For example, when you are working in a network environment, one machine may already have three installed databases and another only has two databases. When you add another database without specifying a port ID, the machines each assign the next available port ID number. This means your new database now has two different port ID numbers. If your network has more than just two machines, the problem is exponentially more difficult.

If the database does not come up on one machine, you cannot merely look at another machine and compare settings. This is because it is possible that one file has an old port ID or the port ID is missing. This happens most often during the initial development phase when most settings are manually entered.

The preferred method for allocating a port ID is to coordinate your port ID assignments with the CWCS database group by sending a request to the `embu-db-interest` alias.

Performing a Quick Integration

The CWCS database property and other files permit flexible configuration for a wide range of application data-storage needs. However, if you plan to follow the basic CWCS database configuration (whether or not you plan to include backup and restore), it is a relatively simple task to create a new database for your application and integrate it with CWCS.

The following procedure describes the minimum set of steps you must perform to get a new database set up and integrated. It summarizes all of the tasks described in detail in the topics under the [“Setting Up a New Database”](#) section on page 11-6.

-
- Step 1** Create the following files (per the guidelines given in the [“Creating the ODBC Database Definition File”](#) section on page 11-7 and the [“Creating the Backup Manifest Files”](#) section on page 11-10):

```
$NMSROOT/databases/dsn/orig/odbc.templorig
$NMSROOT/databases/dsn/orig/odbc.templ
$NMSROOT/databases/dsn/orig/dsn.dborig
$NMSROOT/backup/manifest/suite/database/orig/dsn.txt
```

Where:

- *dsn* is the data source (database) name.
- *suite* is the name of your application or suite. Often, this is the same as the *dsn*. For example: For CWCS, the *suite* and *dsn* are both “cmf”, but for Campus Manager, the *suite* is “campus” and the *dsn* is “ani”.

For example, for a VMS database with “vms” as the *suite* and *dsn*, you would create the following files:

```
$NMSROOT/databases/vms/orig/odbc.templorig
$NMSROOT/databases/vms/orig/odbc.templ
$NMSROOT/databases/vms/orig/vms.dborig
$NMSROOT/backup/manifest/vms/database/orig/vms.txt
```

CISCO CONFIDENTIAL

Step 2 Edit the database backup manifest file (*dsn.txt*) so that it points to your database path, as follows:

```
[ dsn ]
root=$ENV{NMSROOT}/databases/dsn/dsn.db
```

For example, for the VMS database, the database backup manifest contents would look like this:

```
[ vms ]
root=$ENV{NMSROOT}/databases/vms/vms.db
```

Step 3 Add any other databases to the database backup manifest file. For example, RME's *rmeng.txt* file looks like this:

```
[ rme ]
root=$ENV{NMSROOT}/databases/rmeng/rmeng.db
SyslogFirst=$ENV{NMSROOT}/databases/rmeng/SyslogFirst.db
SyslogSecond=$ENV{NMSROOT}/databases/rmeng/SyslogSecond.db
SyslogThird=$ENV{NMSROOT}/databases/rmeng/SyslogThird.db
```

Step 4 Run the following commands to install, register, and create a DbMonitor process for your database:

```
$NMSROOT/objects/db/conf/configureDb.pl action=install dsn=dsn
$NMSROOT/objects/db/conf/configureDb.pl action=reg dsn=dsn dmprefix=<ur_dmprefix>
```

For example, to install and register the VMS database, you would run these commands:

```
$NMSROOT/objects/db/conf/configureDb.pl action=install dsn=vms
$NMSROOT/objects/db/conf/configureDb.pl action=reg dsn=vms dmprefix=<ur_dmprefix>
```



Note If you do not want include your database in the CWCS backup and restore processes, you can simply delete the backup manifest file `$NMSROOT/backup/manifest/suite/database/dsn.txt`. This file is created from `$NMSROOT/backup/manifest/suite/database/orig/dsn.txt` when you run the `configureDb.pl action=install/action=reg` commands.

Using the Sybase Database

The following topics describe some of the typical database tasks you might perform:

- [Before You Begin](#)
- [Setting Up Your Environment](#)
- [Initializing a New Database](#)
- [Creating a New Database](#)
- [Updating the Database Password](#)
- [Starting and Stopping Database Engines](#)
- [Creating and Closing Database Connections](#)
- [Examining the Contents of a Database](#)
- [Backing Up Your Database](#)

CISCO CONFIDENTIAL

For more information, refer to:

- The wrapper classes in `dbservice2`. `DBClient.ExecuteUpdate` and `DBClient.ExecuteSelect` are two database manipulation classes for the update, delete, query and create functions.
- *Sybase Adaptive Server Anywhere Reference Manual*, Chapter 4, “Database Administration Utilities.”
- Sun's JDBC manual.

Before You Begin

When you create a Perl application that interfaces with the database APIs, follow these guidelines:

- Use “my” variables whenever possible—These variables have a true local scope; a local variable in Perl is not truly local as in the C language.
- Always specify “use strict”—This generates errors at compile time for any variables not properly defined in scope and any typos in variables masquerading as null values.
- Always run Perl using `perl -w {your script}`—This generates warnings at runtime for variables that have not been initialized prior to being used.
- Always check for errors from every DBI call.
- Do not confuse DBI with `dbi.pl`—`dbi.pl` contains code specific to the Resource Manager Essentials (RME) database only, and its routines are primarily used in reports. DBI is a general-purpose public domain API contained in `DBI.pm`.

For guidelines when creating JDBC or ODBC applications, refer to the third-party manuals for those interfaces.

Setting Up Your Environment

Before you can initialize your database or run any Sybase utilities, you must set up your environment:

- On Windows platforms, if you have installed **CMF 1.2** or higher, your environment settings have already been created. The only settings you may need to update will be those that apply to any custom databases.
- On Solaris platforms, set the following environment variables:

```
setenv SATMP /tmp/.SQLAnywhere
setenv ASANY $NMSROOT/objects/db
setenv LD_LIBRARY_PATH $NMSROOT/objects/db/lib:$NMSROOT/lib
setenv PATH ${PATH} : $NMSROOT/bin : $NMSROOT/objects/db/bin
```

where `$NMSROOT` is the directory in which the product was installed.

Initializing a New Database

Follow the procedure appropriate for your database platform.

On Windows Platforms

To initialize a database on Windows:

CISCO CONFIDENTIAL

Step 1 Log in as a local administrator and open a DOS window.

Step 2 Enter:

```
cd $NMSROOT/objects/db/win32
```

where *\$NMSROOT* is the directory in which the product was installed.

Step 3 Initialize the database:

```
dbinit -j [-b] [-c] [-p page-size] dbName.db
```

For example, the CWCS database is initialized using this command:

```
dbinit -j -b -c -p 1024 cmf.db
```

On Solaris Platforms

To initialize a database on Solaris:

Step 1 Log in as root.

Step 2 From the command line, set the environment variables (see the “[Setting Up Your Environment](#)” section on page 11-19).

Step 3 Enter:

```
cd $NMSROOT/objects/db/bin
```

where *\$NMSROOT* is the directory in which the product was installed.

Step 4 Initialize the database:

```
dbinit -j [-b] [-c] [-p page-size] dbName.db
```

For example, the CWCS database is initialized using this command:

```
dbinit -j -c -b -p 1024 cmf.db
```

Related Topics

- For more information about the dbinit utility, see the “[dbinit](#)” section on page 11-54.
- See the *Sybase Adaptive Server Anywhere Reference Manual*, Chapter 4, “Database Administration Utilities,” section “The Initialization Utility.”

Creating a New Database

After you have initialized the database, make the following changes to this file:

Step 1 Change the user ID and password. The default database user ID is DBA and default password is SQL.



Warning

If you do not change these values, you will create a security hole.

This procedure is described in the “[Step 1: Change the User ID and Password](#)” section on page 11-21.

CISCO CONFIDENTIAL

- Step 2** Create and populate DbVersion and DbVersionHistory.
This procedure is described in the “[Step 2: Create and Populate DbVersion and DbVersionHistory](#)” section on page 11-21.
- Step 3** Copy the database file to the required directory locations.
This procedure is described in the “[Step 3: Install the Database Files](#)” section on page 11-23.

Related Topics

See the “[Initializing a New Database](#)” section on page 11-19.

Step 1: Change the User ID and Password

Use the changepwd.sql script to change the user ID and password. An SQL script that must run within the Sybase dbisqlc utility, changepwd.sql, changes only the password in the database, *not* the passwords in related configuration files such as obdc.ini and odbc.templorig. A copy of this script is included in the CodeSamples directory on the SDK CD.

Follow the procedure appropriate for your database platform.

On Windows Platforms

To change the user ID and password on Windows:

- Step 1** Log in as a local administrator and open a DOS window.
- Step 2** Enter (on one line):
- ```
dbisqlc -q -c "uid=DBA;pwd=SQL;dbf=newdb.db" read changepwd.sql newuid newpwd
```

**On Solaris Platforms**

To change the user ID and password on Solaris:

- Step 1** Log in as root.
- Step 2** Set the environment variables (see the “[Setting Up Your Environment](#)” section on page 11-19).
- Step 3** Enter (on one line):
- ```
dbisqlc -q -c "uid=DBA;pwd=SQL;dbf=newdb.db" read changepwd.sql newuid newpwd
```

Step 2: Create and Populate DbVersion and DbVersionHistory

Database restore and upgrade utilities require some means of identifying installed device families and the current database version. Two tables track the schema version, dropins and incremental device support:

- DbVersion—This table contains the latest or most recent entries. The schema for this table is:

CISCO CONFIDENTIAL

```

create table DbVersion (
    Componentvarchar(30) not null,
    subComponentvarchar(30) not null
    VersionStringvarchar(20)
    Descriptionvarchar(50)
    InstallDatetimestamp not null default current timestamp,
    Primary key (component, subComponent)
);

```

The information in DbVersion table is updated when IDs are added to the system.

- DbVersionHistory—This table contains a history of the changes to the database. The schema for this table is:

```

create table DbVersionHistory (
    Componentvarchar(40) not null,
    subComponentvarchar(30) not null
    VersionStringvarchar(20)
    Descriptionvarchar(50)
    InstallDatetimestamp,
    Primary key (component, subComponent, InstallDate)
);

```

The DbVersionHistory table is updated with an update/delete trigger on the DbVersion table without requiring applications to insert rows into the DbVersionHistory table.

The content of the DbVersion and DbVersionHistory tables is controlled by individual applications. Although it might not seem very useful when a new application is developed, you can use the Component, subComponent, and VersionString fields to distinguish databases from several application versions.

Follow the procedure appropriate for your platform.

On Windows Platforms

To create and populate the DbVersion and DbVersionHistory tables on Windows:

Step 1 Log in as a local administrator and open a DOS window.

Step 2 Enter:

```
dbisqlc -q -c "uid=newuid;pwd=newpwd;dbf=newdb.db" read dbversion.sql
```

On Solaris Platforms

To create and populate the DbVersion and DbVersionHistory tables on Solaris:

Step 1 Log in as root.

Step 2 Set the environment variables (see the “Setting Up Your Environment” section on page 11-19).

Step 3 Enter:

```
dbisqlc -q -c "uid=newuid;pwd=newpwd;dbf=newdb.db" read dbversion.sql
```

CISCO CONFIDENTIAL**Step 3: Install the Database Files**

After you have initialized the database, changed the user ID and password, populated the DbVersion and DbVersionHistory tables, you are ready to install the database.

To install the database files:

-
- Step 1** Copy the database file to the database directory. This directory contains the working database.
- Database file name: *dsn.db*
 - Location: *\$NMSROOT/databases/suite*
where *\$NMSROOT* is the directory in which the product was installed.
 - Example: *\$NMSROOT/databases/cmfc/cmfc.db*
- Step 2** Copy the database file to the /orig database directory. This directory contains a copy of the original database, which can be used to reinitialize a corrupted database.
- Database file name: *dsn.dborig*
 - Location: *\$NMSROOT/databases/suite/orig*
 - Example: *\$NMSROOT/databases/cmfc/orig/cmfc.dborig*



Note If you are an NMTG developer, just copy the *dsn.dborig* file to the backup (/orig) directory. The automated build processes use this file to create the corresponding .db file on the target system.

Also be sure you have set up the backup manifest files, as explained in [“Creating the Backup Manifest Files” section on page 11-10](#).

- Step 3** Once you have set up the database files, you can use configureDb.pl to install and register database:
- To install the database, run the following command

```
perl configureDb.pl action=install <dsn=database>
```

This command copies the database file from the /orig directory to the runtime directory.
 - To register the database, run the following command:

```
perl configureDb.pl action=reg <dsn=database> <dmprefix=prefix>
```

This command registers the database with the Daemon Manager, including populating the .odbc.ini or Windows odbc registry, updating dmgt.conf or the Windows services registry, and updating the DBServer.properties file.
- Step 4** If needed, you can use the same script to re-install, uninstall, register or unregister the database:
- To reinstall the database, run the following command

```
perl configureDb.pl action=install <dsn=database>
```
 - To uninstall the database, run the following command

```
perl configureDb.pl action=uninstall <dsn=database>
```

This command removes the database file from the runtime directory.
 - To re-register the database, run the following command:

```
perl configureDb.pl action=reg <dsn=database> <dmprefix=prefix>
```
 - To unregister the database, run the following command:

CISCO CONFIDENTIAL

```
perl configureDb.pl action=unreg <dsn=database> <dmprefix=prefix>
```

This command unregisters the database with Daemon Manager.

Related Topics

See the:

- “Restoring a Corrupt Database” section on page 12-18
- “Reinitializing a Database” section on page 11-35.

Updating the Database Password

Use the `dbpasswd.pl` utility at runtime to change the database password. This utility will replace the old password in the `odbc.tmpl` file and populate the new password to:

- The `.odbc.ini` file
- The Windows registry
- The `DbServer.properties` file
- If the entry already exists, any customer-specified Java property file

Note that `dbpasswd.pl` validates submitted passwords. Valid passwords must:

- Have a minimum of five and a maximum of 128 characters.
- Use alphanumeric characters (a-z, A-Z, 0-9) only. No special characters (e.g., #, \$, %) or spaces are allowed.
- Not have a number as the first character.

Follow the procedure appropriate for your platform.

On Windows Platforms

To change the password on Windows:

- Step 1** Log in as a local administrator and open a DOS window.
 - Step 2** Stop the Daemon Manager by entering:

```
net stop crmdmgt
```
 - Step 3** Run the `dbpasswd.pl` utility (see the “`dbpasswd.pl`” section on page 11-57).
 - Step 4** Enter the new password.
 - Step 5** Verify the new password.
 - Step 6** Start the Daemon Manager by entering:

```
net start crmdmgt
```
-

CISCO CONFIDENTIAL

On Solaris Platforms

To change the password on Solaris:

-
- Step 1** Log in as root.
- Step 2** Set the environment variables (see the “[Setting Up Your Environment](#)” section on page 11-19).
- Step 3** Stop the Daemon Manager by entering:
- ```
/etc/init.d/dmgttd stop
```
- Step 4** Run the dbpasswd.pl utility (see the “[dbpasswd.pl](#)” section on page 11-57).
- Step 5** Enter the new password.
- Step 6** Verify the new password.
- Step 7** Start the Daemon Manager by entering:
- ```
/etc/init.d/dmgttd start
```
-

Starting and Stopping Database Engines

The SqlAnywhere embedded API does not provide database auto-start and auto-stop capabilities. Therefore, when Perl DBI applications are run on Solaris platforms, the database must be started explicitly.

The following topics explain how to start and stop a database engine from a Perl application:

- [Starting a Database Engine](#)
- [Stopping a Database Engine](#)

Starting a Database Engine

You can start the database engine from the CWCS Desktop, on Windows or on Solaris. Follow the procedure appropriate for your platform, below.

From the CWCS Desktop

Typically, you would start the database engine (or any other process) from the CWCS desktop:

-
- Step 1** Select **Server Configuration > Administration > Process Management > Start Process**.
The Start Process dialog appears.
- Step 2** Select the name of the database engine from the list box (for example, CmfdBEngine).
- Step 3** Click **Finish**.
-

CISCO CONFIDENTIAL

On Windows Platforms

When creating a new database using a Windows machine, you might have to change the registry entries before starting the database engine. To change the registry entries and start the database engine:

-
- Step 1** Log in as local administrator.
- Step 2** You can use these dialogs in the Control Panel window to add or modify registry entries:
- To start, stop, and configure services (the database engine is registered as a service), select **Start > Settings > Control Panel > Services**.
 - To maintain the ODBC data sources and drivers, select **Start > Settings > Control Panel > ODBC Data Sources > System DSN**.
- Step 3** Or, you can set the Windows registry values directly. The registry entries are located in two places:
- The registry entry used by Windows systems is at:
 My Computer/HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/ Services/CmfDbEngine
 - The registry entries used by the Daemon Manager are located at:
 My Computer/HKEY_LOCAL_MACHINE/SOFTWARE/Cisco/ResourceManager/
 CurrentVersion/Daemons/*
- There is one registry entry for each application that uses the Daemon Manager.
- Step 4** To start the database engine, open a DOS window and enter (all on one line):

```
-x tcpop{HOST=localhost;DOBROADCAST=NO;ServerPort=portID} -m -ti 0 -gm 100 -c 8M -n
yourdbEng $NMSROOT\databases\yourdb\yourdb.db -n yourdbDb
```

where:

- *\$NMSROOT* is the directory in which the product was installed.
- *portID* is the port number assigned to your database.

For example, to start the CWCS database engine, enter:

```
-x tcpop{HOST=localhost;DOBROADCAST=NO;ServerPort=43441} -m -ti 0 -gm 100 -c 8M -n cmfEng
$NMSROOT\databases\cmf\cmf.db -n cmfDb
```

On Solaris Platforms

When creating a new database on a Solaris machine, you can use this procedure during the prototyping phase to start the database engine:

-
- Step 1** Log in as root.
- Step 2** Set the SATMP, SQLANY, and LD_LIBRARY_PATH environment variables (see the “[Setting Up Your Environment](#)” section on page 11-19).
- Step 3** To start the database engine process, enter (all on one line):

```
$NMSROOT/objects/db/bin/dvsrv9 -x tcpip{HOST=localhost;DOBROADCAST=NO;ServerPort=portID} -q -s local0 -m -ti
0 -gm 100 -gc 5 -c 8M -ht -gss 9900 -n yourdbEng $NMSROOT/databases/yourdb/yourdb.db -n yourdbDb
```

where:

- *\$NMSROOT* is the directory in which the product was installed
- *yourdb* is the name of your database engine.
- *portID* is the port number assigned to your database.

CISCO CONFIDENTIAL

For example, to start the CWCS database, enter (all on one line):

```
$NMSROOT/objects/db/bin/dvsrv9 -x tcpip{HOST=localhost;DOBROADCAST=NO;ServerPort=43441} -q -s local0 -m -ti 0
-gm 100 -gc 5 -c 8M -ht -gss 9900 -n cmfEng $NMSROOT/databases/cmf/cmf.db -n cmfDb
```



Note The engine name and database name must be used in the connection string to connect to the specific database.

The first transaction against the database creates the transaction log.

Related Topics

See the [“Connection Strings”](#) section on page 11-5.

Stopping a Database Engine

You can stop a database engine from the CWCS Desktop, on Windows or on Solaris. Follow the procedure appropriate for your platform, below.

From the CWCS Desktop

To stop the database engine from the CWCS desktop:

-
- Step 1** Select **Server Configuration > Administration > Process Management > Stop Process**.
The Stop Process dialog appears.
 - Step 2** Select the name of the database engine from the list box (for example, CmfDbEngine).
 - Step 3** Click **Finish**.
-

On Windows Platforms

To stop the database engine on Windows:

-
- Step 1** Log in as a local administrator and open a DOS window.
 - Step 2** To stop the database engine process, use one of these options:
 - If the SqlAnywhere Console is present, click **Shutdown**.
 - If the database is running as a service, use the Windows service control manager. To access this dialog, select **Start > Control Panel > Services**.
-

**Caution**

DO NOT use the Windows Task Manager unless the process is hanging.

CISCO CONFIDENTIAL**On Solaris Platforms**

To stop the database engine on Solaris:

-
- Step 1** Log in as root.
- Step 2** Set the SATMP, SQLANY, and LD_LIBRARY_PATH environment variables (see the “[Setting Up Your Environment](#)” section on page 11-19).
- Step 3** Stop the database engine process:
- ```
$SQLANY/bin/dbstop -s uid=username;pwd=password;eng=dbEngineName;dbn=dbName
```

**Caution**

Do *not* use the kill command unless the process is hanging.

---

## Creating and Closing Database Connections

Once the database engine is started, there are separate procedures for connecting to and disconnecting from the database. Connection parameters are specified using connection strings. For more information about connection strings, see the “[Connection Strings](#)” section on page 11-5.

The following topics describe how to connect to a database:

- [Connecting to a Database](#)
- [Closing a Database Connection](#)

### Connecting to a Database

You can connect to a database from a Java application, C or C++ application, or Perl script. Follow the procedure appropriate for your application, below.

#### In a Java Application

- Use this call to load the JDBC 5.5 driver, com.sybase.jdbc2.jdbc.SybDriver:

```
Class.forName("com.sybase.jdbc2.jdbc.SybDriver")
```

- The JDBC Sybase component jConnect uses a URL-style syntax:

```
jdbc:sybase:Tds:localhost:42341?SERVICENAME=dbName
where dbName is the name of the database.
```

#### In a C or C++ Application

SQLConnect is the simplest ODBC connection function. It accepts the following connection strings format:

```
"uid=xxx;pwd=yyy;dsn=ddd"
```

SQLDriverConnect can be used to replace SQLConnect. It supports data sources that require more connection information than the arguments in SQLConnect, and data sources that are not defined in the system information. One advantage of using SQLDriverConnect is that we can provide “con=myConnectionName” as part of the connection string to identify the connection. This is useful for debugging and performance analysis.

**CISCO CONFIDENTIAL**

Instead of using either of these functions, however, your applications can use the SQLSecureConnect connection API. SQLSecureConnect acts as a wrapper around SQLDriverConnect to accept and process connection strings that have an encrypted username and password (the custom tokens CWEUID, CWEPWD). It reads the .odbc.ini or registry entry to get either the encrypted or plain text user information. It decodes this information and passes the user ID and password to SQLDriverConnect.



**Note** If encryption is disabled (ENCRYPTION=NO), you can continue to use SQLDriverConnect. SQLSecureConnect, however, supports both encrypted and plain text user IDs and passwords.

If the connection string contains the user ID and password, SQLSecureConnect passes the connection string directly to SQLDriverConnect. If the user ID and password are missing, DSN must be present in the connection string. Other parameters in the connection string are carried over.

SQLSecureConnect uses the following syntax:

```
#include "dbencrypt.h"
SQLRETURN SQLSecureConnect(
 SQLHDBC hdbc,
 SQLHWND hwnd,
 SQLCHAR ODBC FAR * szConnStrIn,
 SQLSMALLINT cbConnStrIn,
 SQLCHAR ODBC FAR * szConnStrOut,
 SQLSMALLINT cbConnStrOutMax,
 SQLSMALLINT ODBC FAR * pcbConnStrOut,
 SQLUSMALLINT fDriverCompletion)
```

(The syntax and semantics of this API are identical to SQLDriverConnect. See the SQLDriverConnect documentation for details.)

For example, the following SQL statements allocate memory for an environment handle, initialize the ODBC call level interface, allocate memory for a connection handle, and use SQLSecureConnect to connect to the database:

```
SQLHENV henv;
SQLHDBC hdbc;
SQLHSTMT hstmt;
SQLRETURN retcode;
char * newconn = (char *)malloc(MAX_DSN_LEN);

/*Allocate environment handle */
retcode = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO) {
 /* Set the ODBC version environment attribute */
 retcode = SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION,
 (void*)SQL_OV_ODBC3, 0);

 if (retcode == SQL_SUCCESS ||
 retcode == SQL_SUCCESS_WITH_INFO) {
 /* Allocate connection handle */
 retcode = SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

 if (retcode == SQL_SUCCESS ||
 retcode == SQL_SUCCESS_WITH_INFO) {
 /* Set login timeout to 5 seconds. */
 SQLSetConnectAttr(hdbc, (void*)SQL_LOGIN_TIMEOUT, 5, 0);

 /* Connect to data source */
 retcode = SQLSecureConnect(
 hdbc,
 0,
```

**CISCO CONFIDENTIAL**

```

(SQLCHAR*)"dsn=cmf",
SQL_NTS,
(SQLCHAR*)newconn,
MAX_DSN_LEN,
&len,
SQL_DRIVER_NOPROMPT);

if (retcode == SQL_SUCCESS
 || retcode == SQL_SUCCESS_WITH_INFO) {
 /* Allocate statement handle */
 retcode = SQLAllocHandle(SQL_HANDLE_STMT,
 hdbc, &hstmt);

 if (retcode == SQL_SUCCESS
 || retcode == SQL_SUCCESS_WITH_INFO) {
 /* Process data */
 ;
 ;
 ;

 SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
 }
 SQLDisconnect(hdbc);
}
SQLFreeHandle(SQL_HANDLE_DBC, hdbc);
}
}

SQLFreeHandle(SQL_HANDLE_ENV, henv);
...

```

**In a Perl Application**

Use `SqlAnywhere` strings (such as `Isql`, `dbstop`, and `dbvalid`):

```
"uid=xxx;pwd=yyy;eng=engName;dbn=dbname;"
```

For example, the following Perl code fragment connects to the CWCS database:

```
use DBI;
my $dbh = DBI->connect ("", 'uid=DBA;pwd=SQL;dsn=cmf', "", 'Sqlany');
```

In this example, DBI will resolve the user ID and password in both encryption modes:

```
use DBI;
my $dbh = DBI->connect ("", 'dsn=cmf', "", 'Sqlany');
```

The last argument is the DBD driver to be loaded. This parameter is ignored on Windows.

**Closing a Database Connection**

You can close a database connection from a Java application, C or C++ application, or Perl script. Follow the procedure appropriate for your application.

**In a Java Application**

Use `dbc.close`.

**In a C or C++ Application**

The following SQL statements close the database connection, release the connection handle, free all memory allocated for the handle, close the ODBC driver, and release all memory associated with the driver:



**CISCO CONFIDENTIAL**

```
SQLDisconnect (dbc);
SQLFreeConnect (dbc);
SQLFreeEnv (env);
```

**In a Perl Application**

```
Use $dbh->disconnect();
```

## Examining the Contents of a Database

You can access your data using the Sybase utility, dbisqlc, or the dbreader utility:

- The Sybase utility, dbisqlc, is more difficult to use but does not require installing CWCS. For information about using dbisqlc, refer to the Sybase documentation.
- The dbreader utility requires that you first install CWCS.

The following topics describe how to use the dbreader utility to access your database:

- [Creating a DSN](#)
- [Accessing Your Data](#)

## Creating a DSN

The dbreader utility uses ODBC to access your database. Before you can use dbreader to access your data, you must create a data source name (DSN) from the ODBC Manager using the steps below.

**Note**

If you have CWCS installed on your desktop, the DSN was created at install time; you can skip this procedure.

**Step 1** Launch the Windows Control Panel.

**Step 2** Click **ODBC Data Source**.

**Step 3** Click **tab-System DSN**.

The name of your database engine should not appear.

**Step 4** Click **Add**.

**Step 5** Click **CiscoWorks Embedded Database**.

**Step 6** Click **Finish**.

The ODBC Configuration for Adaptive Server Anywhere dialog box appears.

**Step 7** Enter the name of your database engine in the **Data Source Name** field.

**Step 8** Click **Login**.

**Step 9** Enter your user ID and password.

**Step 10** Click the **Database** tab.

**Step 11** Complete the following fields:

- Server name: *enginenameEng*
- Database name: *enginenameDb*
- Database file: *\$NMSROOT/databases/enginename/enginename.db*

## CISCO CONFIDENTIAL

**Step 12** Click **OK**.

---

### Accessing Your Data

After you have created a data source name, you can access the contents of your database:

---

**Step 1** Start CiscoWorks and log in with the appropriate user ID and password.

**Step 2** In your browser's address or location bar, enter the following URL:

```
http://hostname:portid/dbreader/dbreader.html
```

The Ad-hoc Retrieval of Database dialog box appears.

**Step 3** Enter the database user ID and password. (Do not confuse them with the CWCS admin user ID and password.)

**Step 4** Enter the database name (for example, cmf, rme, or ani).

**Step 5** To read the database, perform one of the following options:

- Leave **SQL statement to execute** blank, then select **Get Database Tables**. This option retrieves all the Cisco tables in that database. Click on the table name to drill down to the table data.
  - Enter SQL statements in **SQL statement to execute**, then select **Execute SQL Statement**.
- 

### Backing Up Your Database

CWCS provides two different backup options:

- Back up now

To run a backup immediately, select **Server Configuration > Admin > Backup > Back Up Data Now**.

- Scheduled backup

To schedule a backup, select **Server Configuration > Admin > Backup > Schedule Backup**.

When you use either option, all installed application groups are backed up; you are not allowed to select specific application groups.

For more information about using these dialogs, click **Help** at the bottom of the dialog box.

You can also use the backup.pl script to back up the database and all installed application groups. The backup.pl script also uses the backup manifest files to determine which files and directories to back up. For more information on this script, see the [“backup.pl” section on page 12-10](#).

#### Related Topics

See the:

- [“Using the Sybase Database” section on page 11-18](#).
- [“Creating the Backup Manifest Files” section on page 11-10](#).

**CISCO CONFIDENTIAL**

# Debugging and Troubleshooting the Database

The following topics describe how to use the database utilities to troubleshoot database problems:

- [Managing Database Log Files](#)
- [Ensuring Sufficient Temporary Space](#)
- [Optimizing Query Processing](#)
- [Verifying a Database](#)
- [Reinitializing a Database](#)
- [Cleaning Up Other Application Files](#)

For information about backing up and restoring a database, see [Chapter 12, “Using Backup and Restore.”](#)

## Managing Database Log Files

There are two database files that, while transparent to the user, may require attention from the engineer during database development:

- **Transaction log**—Contains a record of the database transactions for this session.  
This file is erased whenever the database engine shutdown process is successful and the database server is running with the `-m` option. When an exception occurs, however, this file remains. You can use the contents of this file to help troubleshoot database problems.
- **Temporary files**—Used for intermediate result sets, and stored in the `$(SATMP)` or `$(TMPDIR)` directory.  
These files are erased during typical database operations. They do require a certain amount of swap space, however, which may be overrun during debugging and testing cycles. When exceptions occur during database development and testing, consider cleaning these directories *only* if you are certain that CWCS is the only application using them.

## Ensuring Sufficient Temporary Space

The Sybase version supplied with this release of CWCS introduced the `temp_space_limit_check` option. When `temp_space_limit_check=on` and there is insufficient temp space for a database connection, the connection will fail. If this option is set `off` and there is insufficient temp space for a connection, the database server will crash.

This option is set `on` for the CWCS database by default. To set this option on for your database, run the SQL statement `set option public.temp_space_limit_check='on'`. To check the option setting, run `select connection_property('temp_space_limit_check')`.

## Optimizing Query Processing

The Sybase option `OPTIMIZATION_GOAL` controls how query processing is optimized. It has two allowed values:

- `first-row`: Returns the first row as quickly as possible.
- `all-rows`: Minimizes the cost of returning the complete result set

**CISCO CONFIDENTIAL**

In previous versions of CWCS, the default setting was `first-row`. In this release, the default setting is `all-rows`.

If you create your CWCS database using older Sybase binaries and rebuilding them to the Sybase 9.x format, you will retain `first-row` as the default `OPTIMIZATION_GOAL` setting. If you create your database file using Sybase 9.x binaries, the default setting will be `all-rows`.

We recommend that you:

1. Determine which `OPTIMIZATION_GOAL` setting your application is using. To do so, run the SQL statement `select connection_property('OPTIMIZATION_GOAL')`.
2. Evaluate the performance of the modules in your application under that setting.
3. Change to the opposite setting and evaluate its performance. To change the setting, run `set option public.OPTIMIZATION_GOAL='all-rows'` or `set option public.OPTIMIZATION_GOAL='first-row'`.
4. Based on your test results, choose whether to change this option setting for your database, as needed.

You can find additional information in the following Sybase documentation:

- [Sybase Adaptive Server Anywhere Database Administration Guide](#): See the section “Optimization\_Goal option” on page 613.
- [Sybase Adaptive Server Anywhere SQL Reference](#): See the “FROM clause” section on page 445. This section discusses the `FASTFIRSTROW` table hint.

## Verifying a Database

Use the `dbvalid` script to validate the integrity of a database. To run `dbvalid`, enter:

```
cd $NMSROOT/databases/dbfile
dbvalid -c uid=$uid;pwd=$pwd;dbf=dbfile
```

where:

- `$NMSROOT` is the directory in which the product was installed.
- `$uid` and `$pwd` are the user ID and password for your database.
- `dbfile` is the database engine name.

The `dbvalid` utility returns one of the following responses:

- No problem.
- One or more tables have been corrupted.  
See the “[Restoring a Corrupt Database](#)” section on page 12-18.
- Cannot bring up the database engine

If you have a log file, try this:

```
rm -f rme.log
dbeng9 -f rme.db
```

This forces the RME database to start up without a transaction log file. Then call `dbvalid` to revalidate the database.

You can also try running the `configureDb/dbvalid` command without passing the entire connection string as an argument.

## CISCO CONFIDENTIAL

### Related Topics

See the:

- “Types of Database Servers” section on page 11-3.
- “dbvalid” section on page 11-60.
- “configureDb.pl” section on page 11-53

## Reinitializing a Database

If the data in a database is totally corrupted or not important, you can copy a clean database from the orig directory over the existing database. You will lose the application data. However, this can be useful if your application database is the only problem the application is having, since you will not need to re-install the application.

When run, dbRestoreOrig.pl prompts for user confirmation, warning that all data will be lost. If your application is using this script internally and you do not want this prompt to appear, add the `opt=y` argument to the dbRestoreOrig.pl call. For example:

```
$NMSROOT/bin/dbRestoreOrig.pl dsn=cmf dmprefix=Cmf opt=y
```

You can also use the call to dbRestoreOrig.pl to clean up application configuration and other data stored in the file system. For details, see the “Cleaning Up Other Application Files” section on page 11-36.

If the data in the database is important, and you have been using a backup framework for regular maintenance, you can use the corresponding restore framework to recover it. For information about the restoring a database from a regularly made backup, see the “Using CWCS Restore” section on page 12-4.

### On Windows Platforms

To reinitialize the database on Windows:

---

**Step 1** Stop the Daemon Manager by entering:

```
net stop crmdmgtd
```

**Step 2** Enter on one line:

```
$NMSROOT/bin/dbRestoreOrig.pl dsn=dsn dmprefix=dmprefix
```

where:

- *NMSROOT* is the directory in which the product was installed.
  - *dbn* is your database name.
  - *dmprefix* is the prefix registered for this database in the CWCS Daemon Manager.
- 

### On Solaris Platforms

To reinitialize the database on Solaris:

---

**Step 1** Stop the Daemon Manager by entering:

```
/etc/init.d/dmgtd stop
```

**CISCO CONFIDENTIAL****Step 2** Enter:

```
NMSROOT/bin/dbRestoreOrig.pl dsn=$dbn dmprefix=$dmprefix
```

where:

- *NMSROOT* is the directory in which the product was installed.
  - *dsn* is your database name.
  - *dmprefix* is the prefix registered for this database in the CWCS Daemon Manager.
- 

## Cleaning Up Other Application Files

The script `dbRestoreOrig.pl` (see the “[dbRestoreOrig](#)” section on page 11-59) lets you reinitialize a corrupt database without touching the application’s file system. For example, if MyApp has configuration files, logs, archives, or images stored outside of the database, all of these files will still be in the same locations after you reinitialize MyApp’s database using `dbRestoreOrig.pl`.

If you want to eliminate these file system leftovers at the same time you reinitialize the database, you can create a script to do so and get `dbRestoreOrig.pl` to execute it, as follows:

1. Define a script to be executed after the database is reinitialized.  
The script must implement a `Cleanup::doCleanup()` function that meets your requirements, and should return zero for success and a non-zero value for failure (see [Example 11-1](#)).
2. Store the script as `Cleanup.pm` in `$NMSROOT/databases/dsn/scripts/` (where *dsn* is your application’s data source name).
3. Run `dbRestoreOrig.pl` with the appropriate *dsn*. The utility will restore the orig database.
4. Before exiting, `dbRestoreOrig.pl` will look for the `Cleanup.pm` file. If the utility finds this file, it will execute the `Cleanup.pm` file’s `Cleanup::doCleanup()` as its last act.

### **Example 11-1** Cleanup.pm Script

---

```
sub doCleanup{
 LogError("Inside CMF Cleanup\n") ;
 # add your code here
 # return 0 for success and non-zero for failure
 return 0 ;
}
1
```

---

**CISCO CONFIDENTIAL**

# Database API Command Reference

The following topics describe how to integrate your database into CWCS:

- [Enabling the CWCS Database Engine](#)
- [Using JDBC API Wrappers](#)
- [Using CWCS Perl APIs](#)
- [Using the Database Utilities](#)

## Enabling the CWCS Database Engine

Your custom database runs independently from the CWCS database engine. When using CWCS, however, you will also be using the CWCS database engine.

The database engine is part of the Common Services, but it is not enabled by default. You must enable it before you can use it. If your application requires services from the CWCS database engine, remember to register for this service at installation.

For instructions, refer to the [“Registering for CWCS Services” section on page 5-4](#). If you prefer to request services after installation, refer to the [“Enabling New Service Bundles from the Command Line” section on page 5-5](#).

**Note**

---

If you have *installed* the CWCS database but have not *enabled* it, you will *not* have access to any ODBC or JDBC commands. Perl also uses ODBC commands, so it will not work either.

---

## Compiling and Running a Database

When you are developing new applications, remember that:

- Client and client-server Java classes are stored in `$NMSROOT/www/classpath`
- Server-only Java classes are stored in `$NMSROOT/lib/classpath`

Use the procedure appropriate for your platform.

### On Windows Platforms

To compile an application that accesses a database on Windows, enter:

```
$dev:\enm_jdk\jdk1.2.1\NT\bin\javac -classpath
$NMSROOT\lib\classpath;$NMSROOT\www\classpath appname.java
```

where:

- `$dev` is the location of the JDK.
- `$NMSROOT` is the directory in which the product is installed.
- `appname` is the name of your application

To run an application that accesses a database on Windows, enter:

```
$NMSROOT\lib\jre2\bin\java -classpath
.;$NMSROOT\www\classpath;$NMSROOT\lib\classpath test
```

**CISCO CONFIDENTIAL****On Solaris Platforms**

To compile an application that accesses a database on Solaris, enter (all on one line):

```
java -classpath .:$NMSROOT/www/classpath:$NMSROOT/lib/classpath testappContent-Type:
text/plain; charset="us-ascii"
Content-Disposition: attachment; filename="compile.sh"
```

where *\$NMSROOT* is the directory in which the product was installed.

To run an application that accesses a database on Solaris, enter (all on one line):

```
javac -classpath
.:$NMSROOT/lib/classpath:$NMSROOT/www/classpath:$NMSROOT/lib/jre/lib/rt.jar testapp.java
```

**Related Topics**

See:

- [Chapter 3, “Understanding the CWCS Directory Structure.”](#)
- The “[Understanding the Java Application Launch Process](#)” section on page 4-1.

**Code Samples**

The following topics contain assorted code samples that illustrate various database tasks:

- [Using Java to Read a Database](#)
- [Using ODBC to Access a Table](#)
- [Using Perl to Access a Database](#)

**Using Java to Read a Database**

The following example shows how to use these JDBC API wrappers from the *dbservice2* package:

- *DBClient*—Connect a database engine and perform database-level and SQL statement-level operations.
- *DBResult*—Examine the query results.
- *DBException*—Handle exception cases.

For more information about these wrappers, see the “[Using JDBC API Wrappers](#)” section on page 11-41.

**Example 11-2 Using Java to Read a Database**


---

```
import java.io.*;
import java.util.*;
import com.cisco.nm.cmf.dbservice2.*;
import java.sql.*;

public class testapp {
 static DBClient dbc = null;
 static DBResult dbr1 = null; // data base results from select and update operations
 static Vector row;

 public static void main(String args[]) {
 System.out.println("test application for DB");
 }
}
```



**CISCO CONFIDENTIAL**

```

try {
 dbc = new DBClient("testapp", "rme", 1);
 //appName, debugLevel (1 means turn on debug messages)
 // gets the URL from the property file and connects to the database.
 // Also creates an SQL statement handle.
} catch (DBException ex) {
 System.out.println("Error Message: " + ex.getMessage());
 if (ex.isSqlError()) {
 System.out.println("SQL ERROR CODE: " + ex.getSqlErrorCode());
 System.out.println("SQL STATE: " + ex.getSqlErrorCode());
 }
 System.out.println("test application failed at: ");
 ex.printStackTrace();
 return;
} catch (ClassNotFoundException ex) {
 System.out.println("Error Message: " + ex.getMessage());
 System.out.println("test application failed at: ");
 ex.printStackTrace();
 return;
}
}
try {
 dbr1 = dbc.executeSelect("select * from dbversion");
 if (dbr1 == null) {
 System.out.println("no data");
 } else {
 dbr1.moveToFirst();

 int count = 0;
 String Component;
 String SubComponent;
 String VersionString;
 String Description;
 String InstallDate;
 while ((row = dbr1.getRow()) != null) { //iterate through all the test results
 Component = row.elementAt(0).toString();
 SubComponent = row.elementAt(1).toString();
 VersionString = row.elementAt(2).toString();
 Description = row.elementAt(3).toString();
 InstallDate = row.elementAt(4).toString();
 System.out.println(Component + " " + SubComponent + " "
 + VersionString + " " + Description + " "
 + InstallDate + "\n\n");
 count++;
 }
 }
} catch (DBException ex) {
 System.out.println("Error Message: " + ex.getMessage());
 if (ex.isSqlError()) {
 System.out.println("SQL ERROR CODE: " + ex.getSqlErrorCode());
 System.out.println("SQL STATE: " + ex.getSqlErrorCode());
 }
 System.out.println("test application failed at: ");
 ex.printStackTrace();
}
try {
 dbc.close();
} catch (DBException ex) {
 System.out.println("Error Message: " + ex.getMessage());
 if (ex.isSqlError()) {
 System.out.println("SQL ERROR CODE: " + ex.getSqlErrorCode());
 System.out.println("SQL STATE: " + ex.getSqlErrorCode());
 }
}
System.out.println("test application failed at: ");

```

**CISCO CONFIDENTIAL**

```

 ex.printStackTrace();
 }
 System.out.println("test application finished. ");
}
}

```

**Using ODBC to Access a Table**

The following code fragment shows how to use ODBC calls to access a device group table.

**Example 11-3 Using ODBC Calls to Access a Device Group Table**

```

strcpy((char *)connStr, "uid=DBA;pwd=SQL;dsn=rme");
if (SQLAllocEnv(&henv) != SQL_SUCCESS) { /* handle error */
if (SQLAllocConnect(henv, &hdbc) != SQL_SUCCESS) {
 print_error(); exit(1); }
rc = SQLSecureConnect(hdbc, 0, connStr, SQL_NTS,
 outBug, 256, &outBufLen, SQL_DRIVER_NOPROMPT);
if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) { ... }

/* read and print from dev_group table */
rc = SQLAllocStmt(hdbc, &hstmt);
if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO) { ... }
sqlStr = (UCHAR *) "select * from dev_group";
if (SQLExecDirect (hstmt, sqlStr, SQL_NTS) != SQL_SUCCESS) { ... }
SQLNumResultCols(hstmt, &nresultcols);

for (i = 0; i < nresultcols; i++) {
 SQLDescribeCol(hstmt, i + 1, colname, (SQLWORD) sizeof(colname),
 ...
}

```

**Using Perl to Access a Database**

The following code fragment shows how to use Perl to fetch data from the database.

**Example 11-4 Using Perl to Access a Database**

```

use CRM;
use lib "$ENV{NMSROOT}/lib/perl/install";
use InstallUtility;
use lib "$ENV{NMSROOT}/lib/perl/db";
use Cisco::DbUtils;
use dbinternal;

my $dbh;
my $cur;
my $dsn = "cmf";
my @data;
my $Component, $SubComponent, $VersionString, $Description, $InstallDate;
$dbh = &dbinternal::connect("dsn=$dsn");
if (!defined($dbh)) {

```

**CISCO CONFIDENTIAL**

```

 print "\n\n ERROR testdb.pl:Couldn't connect to the database $dsn\n";
 return 0;
}

$cur = $dbh->prepare("SELECT * from DbVersion");
if($cur) {
 $cur->execute();
 while (@data = $cur->fetchrow) {
 ($Component,$SubComponent,$VersionString,$Description,$InstallDate) = @data;
 print "$Component,$SubComponent,$VersionString,$Description,$InstallDate \n";
 $cnt++;
 }
 $cur->finish;
} else {
 print STDERR "Unable to select items from DB :$DBI::errstr";
}

$dbh->disconnect;
1;

```

---

## Using JDBC API Wrappers

The `dbservice2` package contains six classes. You can write a JDBC application by referencing three of them:

- `DBClient` allows you to connect to a database engine and perform database-level and SQL statement-level operations.
- `DBResult` lets you examine the query results.
- `DBException` handles exception cases.

We strongly recommend that you use the `DBUtil.getConnection(java.lang.String dbName)` API to get a `java.sql.Connection` object. While it is still supported, the existing `DBConnection` class (see the [“DBConnection” section on page 11-44](#)) is being deprecated.

### DBClient

This topic describes the `DBClient` constructor and its public methods.

#### DBClient Constructor Summary

---

```
public DBClient (String dbName) throws ClassNotFoundException,
DBException.
```

Same as (“noAppName”, dbName, `DBUtil.getServiceProperties()`, 0);

This class is a wrapper for the `java.sql.Connection` and `java.sql.Statement` classes. It extracts JDBC connection information from the `DBServer.properties` file and creates a connection. It also creates `java.sql.Statement` objects so the application can perform the `java.sql.Statement` operation.

---

**CISCO CONFIDENTIAL****Table 11-3 DBClient Method Summary**

| Returns                               | Syntax and Description                                                                                                                                                                                                                                                                           |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| synchronized<br>void                  | <code>close ()</code> throws <code>DBException</code><br>A wrapper for <code>Connection.close</code> . It also closes the statement it owns.                                                                                                                                                     |
| synchronized<br>void                  | <code>commit ()</code> throws <code>DBException</code><br>A wrapper for <code>Connection.commit</code> .                                                                                                                                                                                         |
| void                                  | <code>createPreparedStatement (String sql)</code> throws <code>DBException</code><br>Creates a <code>PreparedStatement</code> object.                                                                                                                                                            |
| void                                  | <code>disableRetryOnLockedRow ()</code><br>Disallows retry on locked row.                                                                                                                                                                                                                        |
| void                                  | <code>enableRetryOnLockedRow (Integer maxTryCount, Long tryInterval)</code><br>Allows retry on locked row. If <code>RetryOnLockedRow</code> is true, an SQL statement will try up to <code>maxTryCount</code> or until the statement is executed successfully.                                   |
| synchronized<br><code>DBResult</code> | <code>executeSelect (String sql)</code> throws <code>DBException</code><br>A wrapper for <code>Statement.executeQuery</code> . Tries to rebuild a connection if the old connection is dropped. If it fails, it retries up to <code>MaxTryCount</code> if <code>RetryOnLockedRow</code> is true.  |
| synchronized<br><code>DBResult</code> | <code>executeUpdate (String sql)</code> throws <code>DBException</code><br>A wrapper for <code>Statement.executeUpdate</code> . Tries to rebuild a connection if the old connection is dropped. If it fails, it retries up to <code>MaxTryCount</code> if <code>RetryOnLockedRow</code> is true. |
| synchronized int                      | <code>getDebugLevel ()</code> throws <code>DBException</code><br>Gets the debug level.                                                                                                                                                                                                           |
| void                                  | <code>reOpenStaleConnection (Boolean reOpenStaleConnection)</code> throws <code>DBException</code><br>Reconnects to a database if the connection drops for any reason by setting the <code>reOpenStaleConnection</code> flag to true.                                                            |
| synchronized<br>void                  | <code>rollback ()</code> throws <code>DBException</code><br>A wrapper for <code>Connection.rollback</code> .                                                                                                                                                                                     |
| synchronized<br>void                  | <code>setAutoCommit (boolean autoCommit)</code> throws <code>DBException</code><br>A wrapper for <code>Connection.setAutoCommit</code> .                                                                                                                                                         |
| synchronized<br>void                  | <code>setDebugLevel (int debugLevel)</code> throws <code>DBException</code><br>Sets the debug level.                                                                                                                                                                                             |
| synchronized<br>void                  | <code>setTransactionIsolation (int ti)</code> throws <code>DBException</code><br>A wrapper for <code>Connection.setTransactionIsolation</code> .                                                                                                                                                 |
| synchronized<br>void                  | <code>setTransactionIsolation (Integer ti)</code> throws <code>DBException</code><br>A wrapper for <code>Connection.setTransactionIsolation</code> .                                                                                                                                             |

**DBResult**

The `DBResult` class is a wrapper for `ResultSet`. This topic provides a summary of its constructors and public methods.

**DBResult Constructor Summaries**


---

```
public DBResult (int rowsAffected)
A wrapper for ResultSet.
```

---

**CISCO CONFIDENTIAL**


---

```
public DBResult (ResultSet resultSet) throws SQLException,
DBException
```

Copies and stores data in private storage.

---

```
public DBResult (ResultSet resultSet, int storageType) throws
SQLException, DBException
```

Copies and stores data in private storage.

---

**Table 11-4 DBResult Method Summary**

| Returns                | Syntax and Description                                                                                                                                                                |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int                    | <code>getAffectedRows()</code> throws <code>DBException</code><br>Returns the number of updated records after an update operation.                                                    |
| int                    | <code>getColCount()</code> throws <code>DBException</code><br>Returns column count.                                                                                                   |
| int                    | <code>getColDisplaySize (int col)</code> throws <code>DBException</code><br>Given a column position, returns a column display size by fetching column metadata stored in this object. |
| int                    | <code>getColType (int col)</code> throws <code>DBException</code><br>Given a column position, returns a column type by fetching column metadata stored in this object.                |
| Vector                 | <code>getResultVector()</code> throws <code>DBException</code><br>Returns all records.                                                                                                |
| synchronized<br>Vector | <code>getRow()</code><br>Moves the cursor down and fetches the next record.                                                                                                           |
| int                    | <code>getRowCount()</code> throws <code>DBException</code><br>Returns the number of records after a fetch operation.                                                                  |
| void                   | <code>toFirst()</code><br>Moves the cursor to the first row in result set.                                                                                                            |
| void                   | <code>toPrintWriter (PrintWriter pw)</code> throws <code>IOException</code><br>Sends formatted result set to an output stream.                                                        |
| String                 | <code>toString ()</code><br>Converts the object to string with “\t” as element delimiter and “\r\n” as line delimiter.                                                                |
| String                 | <code>toString (String elementDelimiter, String lineDelimiter, String beginDelimiter, String endDelimiter)</code><br>Converts the object to string-supplied delimiters.               |

**Class DBUtil**

The `DBUtil` class loads the JDBC property file. [Table 11-5](#) contains a summary of its public methods.

**Table 11-5 DBUtil Method Summary**

| Returns              | Syntax and Description                                                                                                                                                                                                                     |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| static void          | <code>debugPrint (String app, String catg, String message)</code><br>Prints formatted message with arguments.                                                                                                                              |
| static<br>String     | <code>extractStackTrace (Exception ex)</code><br>Prints stack.                                                                                                                                                                             |
| static<br>Properties | <code>getDBServiceProperties()</code> throws <code>DBException</code><br>Returns an object of class <code>Properties</code> with a pair of (name, value) as ( <code>_DBS__PROPS</code> , “com/cisco/nm/cmfdbservice/DBServer.properties”). |

**CISCO CONFIDENTIAL****Table 11-5 DBUtil Method Summary**

| Returns           | Syntax and Description                                                                                                                                                                                             |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| static String     | <code>getPropertiesFileName()</code><br>Returns the value of environment variable BG_DBPARAMS if it is defined. Else it returns <code>bgdbparam.ini</code> .                                                       |
| static Properties | <code>loadPropertiesFromFile (String propertiesFile)</code> throws <code>IOException</code> , <code>FileNotFoundException</code><br>Loads the class <code>Properties</code> from <code>DBConst._DBS_PROPS</code> . |
| static void       | <code>printExceptionDetails (Exception e)</code><br>Prints contents of the exception object.                                                                                                                       |
| static void       | <code>printExecuteDebugStmt (long startTime, long stopTime, String sql)</code><br>Prints formatted message with arguments.                                                                                         |

In CS3.0 SP2, two flavors of the new API, `executeSqlStmt()` have been created.

One takes default delimiter (;), and the other takes the customized delimiter.

**Flavor A**

The `executeSqlStmt()` API is used to execute a bunch of SQL statements separated by the customized delimiter.

@param `dsn`—The DSN for the database

@param `schemaFileName`—The file containing the schema to create the tables. It uses `getResourceAsStream` to locate the file. Hence this file must be present in the classpath.

@param `component`—The name of the component/product eg. Kilner/Campus

@param `subComponent`—The name of the subcomponent eg. OGS/DCR

@param `versionString`—The version number of the release eg: 1.0/1.1

@param `description`—A brief description of the module/schema

@param `delimiter`—Delimiter for the SQL Statements.

@return—Throws an exception if the operation fails for any reason

**Flavor B:**

The `executeSqlStmt()` API is used to execute a bunch of SQL statements separated by the default (;) delimiter. This API takes all the parameters specified in the Flavor A, except the delimiter. Here, the default delimiter is semicolon (;). This API does not support to create procedures

**DBConnection**

One instance is constructed in the `DBClient` construct method. An application developer can choose to skip this section.

The `DBConnection` construct gets the URL from the JDBC property file and establishes a connection to a database engine (default = rme). This topic summarizes the constructor and its public methods.

**DBConnection Constructor Summary**


---

```
public DBConnection (String dbName) throws SQLException, DBException,
FileNotFoundException, IOException, ClassNotFoundException
Builds a connection with dbName.
```

---

**CISCO CONFIDENTIAL****Table 11-6 DBConnection Method Summary**

| Returns           | Syntax and Description                                                                                         |
|-------------------|----------------------------------------------------------------------------------------------------------------|
| void              | activate (String client)<br>Activates a client by adding the client name to the internal hash table.           |
| void              | clearWarnings() throws SQLException<br>A wrapper for Connection.clearWarnings                                  |
| void              | close() throws SQLException<br>A wrapper for Connection::close();                                              |
| void              | commit() throws SQLException<br>A wrapper for Connection::commit();                                            |
| Statement         | createStatement() throws SQLException<br>A wrapper for connection::createStatement();                          |
| boolean           | getAutoCommit() throws SQLException<br>A wrapper for Connection::getAutoCommit();                              |
| String            | getCatalog() throws SQLException<br>A wrapper for Connection.getCatalog                                        |
| String            | getClientName()<br>Returns a client name.                                                                      |
| int               | getDebugLevel() {<br>Gets the Cisco debug level.                                                               |
| DatabaseMetaData  | getMetaData() throws SQLException<br>A wrapper for Connection::getMetaData();                                  |
| int               | getTransactionIsolation() throws SQLException<br>A wrapper for Connection.getTransactionIsolation              |
| SQLWarning        | getWarnings() throws SQLException<br>A wrapper for Connection.getWarnings                                      |
| void              | inactivate()<br>Deactivates a client by removing the client name from the internal hash table.                 |
| boolean           | isClosed() throws SQLException<br>A wrapper for Connection:: isClosed();                                       |
| boolean           | isReadOnly() throws SQLException<br>A wrapper for Connection.isReadOnly                                        |
| String            | nativeSQL (String sql) throws SQLException<br>A wrapper for Connection: nativeSQL(sql);                        |
| CallableStatement | prepareCall (String sql) throws SQLException<br>A wrapper for Connection.prepareCall(sql);                     |
| PreparedStatement | prepareStatement (String sql) throws SQLException<br>A wrapper for connection::prepareStatement(sql);          |
| void              | rollback() throws SQLException<br>A wrapper for Connection::rollback();                                        |
| void              | setAutoCommit (boolean autoCommit) throws SQLException<br>A wrapper for Connection::setAutoCommit(autoCommit); |
| void              | setCatalog (String catalog) throws SQLException<br>A wrapper for Connection.setCatalog                         |
| void              | setClientName (String client)<br>Sets a client name.                                                           |

**CISCO CONFIDENTIAL****Table 11-6** DBConnection Method Summary (continued)

| Returns | Syntax and Description                                                                                                                          |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| void    | <code>setDebugLevel (int debugLevel)</code><br>Sets the Cisco debug level.                                                                      |
| void    | <code>setReadOnly (boolean readOnly)</code> throws <code>SQLException</code><br>A wrapper for <code>Connection.setReadOnly</code> .             |
| void    | <code>setTransactionIsolation (int ti)</code> throws <code>SQLException</code><br>A wrapper for <code>Connection.setTransactionIsolation</code> |

## Using CWCS Perl APIs

The database APIs are implemented in Perl for portability between platforms. Because these APIs are primarily used by install and database administration scripts such as backup and restore, they will not be implemented in other languages.

### Programming Tips for Perl APIs

- All applications that use these APIs must include this line at the beginning of their file:  
`use Cisco::DbUtils;`
- These APIs return 0 for success and 1 for error. The variable `Cisco::Dbutil::errstr` is null if the return value is 0 and contains the error string if the return value is 1.
- For more information about connection strings, see the “[Connection Strings](#)” section on page 11-5.

### Perl API Summaries

The following tables summarize the Perl APIs.

**Table 11-7** Database Process and File Management Perl APIs

| Returns | Syntax and Description                                                                                                                                                           |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int     | <code>CheckDb (\$connstr);</code><br>Checks if the specified database accepts connections. This is done by opening a connection to the database.                                 |
| int     | <code>StartDb (\$connstr, \$cacheSize, \$timeout, \$numRetries)</code><br>Starts the database engine as a process on the specified database.                                     |
| int     | <code>StopDb (\$connstr, \$timeout, \$numRetries);</code><br>Stops the specified database engine process. Programs must be sure there are no active connections to the database. |

**Table 11-8** Miscellaneous Perl APIs

| Returns | Syntax and Description                                                                                                                                                                                                                                             |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int     | <code>addManifestFiles ("SUITE=xxx;SWITCH", \@parm2, \\$manifestPath);</code><br>Creates the manifest files used by the backup, restore, and move database utilities. This routine works in one of two ways, depending on the SWITCH field of the first parameter. |
| int     | <code>check_create (\$Dir)</code><br>Verifies the existence of the specified directory and creates it if it does not exist.                                                                                                                                        |



**CISCO CONFIDENTIAL****Table 11-8** Miscellaneous Perl APIs (continued)

| Returns | Syntax and Description                                                                                                                                                                                                         |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int     | <code>deleteDbVersionData (\$dbh, \$component, \$subComponent);</code><br>Using the database handle and the primary keys, this routine deletes a row from the DbVersion table.                                                 |
| int     | <code>deleteManifestFiles ("SUITE=xxx;SWITCH", \ \$manifestPath);</code><br>Deletes the datafiles.txt or the appropriate dbfiles.txt file.                                                                                     |
| int     | <code>getDbVersionData (\$dbh, \$comp, \$subcomp, \$rhDdata);</code><br>Using the database handle and the primary keys, this routine retrieves a row from the DbVersion table. The complete row entry is returned in \$rhData. |
| int     | <code>getManifestFiles ("SUITE=xxx;SWITCH", \@parm2, \ \$manifestPath);</code><br>Retrieves the contents of the datafiles.txt or dbfiles.txt file.                                                                             |
| int     | <code>setDbVersionData (\$dbh, \$rhData);</code><br>Writes a row to the DbVersion table.                                                                                                                                       |
| int     | <code>unloadDbVersionData (\$dbh, \$file);</code><br>Unloads the contents of the DbVersion table into the specified file.                                                                                                      |

**addManifestFiles**

```
$ret = addManifestFiles ("SUITE=xxx;SWITCH", \@parm2, \ $manifestPath);
```

Creates the manifest files used by the backup, restore, and move database utilities. This routine works in one of two ways, depending on the SWITCH field of the first parameter.

**Input Arguments**

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SUITE          | Name for a group of applications. Used for backup and restore purposes.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| SWITCH         | <ul style="list-style-type: none"> <li>APP—Application directory name. Indicates the location of the datafiles.txt file.</li> <li>DSN—The ODBC data source name of the database. Indicates the location of the dbfiles.txt file.</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| @parm2         | <ul style="list-style-type: none"> <li>If SWITCH = "APP=xxx" then @parm2 is a pointer to a list of paths required by the application for backup.<br/>In this case, the routine creates a file called datafiles.txt in the directory <i>\$NMSROOT/backup/manifest/suite/app</i>, where <i>\$NMSROOT</i> is the directory in which the product was installed.<br/>The contents of the file will be the contents of @parm2 (the list of paths containing files to be backed up).</li> <li>If SWITCH = "DSN=xxx" then @parm2 is a pointer to an array of database file paths. In this case, the routine creates a file called dbfiles.txt in <i>\$NMSROOT/backup/manifest/suite/database</i>. The contents of the file will be the DSN value followed by the contents of @parm2 (the list of database files).</li> </ul> |
| \$manifestPath | The complete path of the manifest file. This path information is used to register the file with the UNIX packaging mechanisms.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

**CISCO CONFIDENTIAL****check\_create**

```
$ret = check_create ($Dir);
```

Verifies the existence of the specified directory and creates it if it does not exist.

**Input Arguments**

`$Dir` The name of the directory.

**Return Values**

0 Directory exists or was created successfully.

1 Directory cannot be accessed or failed to create directory.

**checkDb**

```
$ret = CheckDb ($connString);
```

Checks to see if the specified database accepts connections. This is done by opening a connection to the database. If the connection is opened (the routine is successful), the connection is immediately closed.

**Input Arguments**

`$connString` **SQL style:** `ENG=xx;DBN=xx;UID=xx;PWD=xx`  
 where ENG, DBN, UID, PWD are the database engine name, database name, database user ID and password.  
**ODBC style:** `DSN=xx`  
 where the engine corresponding to the ODBC data source `xx` is checked.

**Return Values**

0 Connection accepted.

1 Connection refused.

**deleteDbVersionData**

```
$ret = deleteDbVersionData ($dbh, $component, $subComponent);
```

Deletes a row, defined by the database handle and primary keys, from the database.

**CISCO CONFIDENTIAL****Input Arguments**

|                             |                                                                       |
|-----------------------------|-----------------------------------------------------------------------|
| <code>\$dbh</code>          | Database handle.                                                      |
| <code>\$component</code>    | A primary key into the Schema Version tracking table in the database. |
| <code>\$subcomponent</code> | A primary key into the Schema Version tracking table in the database. |

**deleteManifestFiles**

```
$ret = deleteManifestFiles("SUITE=xxx;SWITCH", \ $manifestPath);
```

Deletes the manifest files used by the backup, restore and move database utilities. This routine works in one of two ways, depending on the SWITCH field of the first parameter.

**Input Arguments**

|                             |                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>SUITE</code>          | Name for a group of applications. Used for backup and restore purposes.                                                                                                                                                                                                                                                                                                                        |
| <code>SWITCH</code>         | <ul style="list-style-type: none"> <li>If SWITCH = "DSN=yyy" then the dbfiles.txt file in the <code>\$NMSROOT/backup/manifest/suite/database</code> is deleted. <code>\$NMSROOT</code> is the directory in which the product was installed.</li> <li>If SWITCH = "APP=yyy" then the datafiles.txtfile in the directory <code>\$NMSROOT/backup/manifest/suite/app</code> is deleted.</li> </ul> |
| <code>\$manifestPath</code> | The complete path of the manifest file. This path information is used to register the file with the UNIX packaging mechanisms.                                                                                                                                                                                                                                                                 |

**getDbVersionData**

```
$ret = getDbVersionData ($dbh, $comp, $subcomp, $rhDdata);
```

Returns the complete row entry in the Schema Version tracking table, identified by the database handle and primary keys, in a hash table.

**Input Arguments**

|                    |                  |
|--------------------|------------------|
| <code>\$dbh</code> | Database handle. |
|--------------------|------------------|

**CISCO CONFIDENTIAL**

|           |                                                                                    |
|-----------|------------------------------------------------------------------------------------|
| \$comp    | Component—a primary key into the Schema Version tracking table in the database.    |
| \$subcomp | Subcomponent—a primary key into the Schema Version tracking table in the database. |

**Output Arguments**

|           |                                                                 |
|-----------|-----------------------------------------------------------------|
| \$rhDdata | Pointer to a hash table containing column name and value pairs. |
|-----------|-----------------------------------------------------------------|

**Example**

```
$dbh = DBI->connect('DSN=xx', undef, undef, 'Sqlany');
$ret = getDbVersionData($dbh, 'Main', 'Baseline', $rhData);
```

where \$rhData is a pointer to a hash whose definition looks like this:

```
$rhData = {'Component' => 'Main', 'SubComponent' => 'Baseline',
 'VersionString' => '208',
 'Description' => 'Database Schema Model Version',
 'InstallDate' => '08/21/1998'};
```

**getManifestFiles**

```
$ret = getManifestFiles ("SUITE=xxx;SWITCH", \@parm2, \$manifestPath);
```

Places the contents of either the datafiles.txt or the dbfiles.txt file into an array. This routine works in one of two ways, depending on the SWITCH field of the first parameter.

**Input Arguments**

|                |                                                                                                                                                                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SUITE          | Name for a group of applications. Used for backup and restore purposes.                                                                                                                                                                                                                                  |
| SWITCH         | <ul style="list-style-type: none"> <li>• APP—Application directory name. Indicates the location of the datafiles.txt file.</li> <li>• DSN—The ODBC data source name of the database. Indicates the location of the dbfiles.txt file.</li> </ul>                                                          |
| @parm2         | <ul style="list-style-type: none"> <li>• If SWITCH = “APP=xxx” then @parm2 contains the contents of the datafiles.txt file, which is the list of file paths containing the files to be backed up.</li> <li>• If SWITCH = “DSN=yyy” then @parm2 contains the contents of the dbfiles.txt file.</li> </ul> |
| \$manifestPath | The complete path of the manifest file. This path information is used to register the file with the UNIX packaging mechanisms.                                                                                                                                                                           |

**CISCO CONFIDENTIAL****setDbVersionData**

```
$ret = setDbVersionData ($dbh, $rhData);
```

Adds or modifies a row in the DbVersion table.

**Input Arguments**

|          |                                             |
|----------|---------------------------------------------|
| \$dbh    | Database handle.                            |
| \$rhData | Contains the row to be inserted or updated. |

**StartDb**

```
$ret = StartDb ($connString, $cacheSize, $timeout, $numRetries);
```

Starts the database specified in the connection string. The connString argument must have dsn=xxx component.

**Input Arguments**

|              |                                                                                                                                                                                                                                                                                |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| \$connString | <p><b>SQL style:</b> “ENG=xx;DBN=xx,UID=xx;PWD=xx”<br/>where ENG, DBN, UID, PWD are the database engine name, database name, database user ID, and password.</p> <p><b>ODBC style:</b> “DSN=xx”.<br/>where the engine corresponding to the ODBC data source xx is stopped.</p> |
| \$cacheSize  | Size (in megabytes) of the database engine. Default = 16M.                                                                                                                                                                                                                     |
| \$timeout    | Number of seconds the routine waits for the database to start on each try. Default = 5 seconds.                                                                                                                                                                                |
| \$numRetries | Number of times the routine attempts to start the database before returning an error. Default = 10 retries.                                                                                                                                                                    |

**Example**

This example starts the database with a cache size of 16M.

```
$ret = StartDb("eng=upgrade; dbn=Essentials; dbf=/opt/CSCOpX/objects/db/px.db; uid=sa;
pwd=c2Ky2k", "16");
```

**StopDb**

```
$ret = StopDb ($connString, $timeout, $numRetries);
```

Stops the specified database engine process. Programs must be sure there are no active connections to the database.

**CISCO CONFIDENTIAL****Input Arguments**

|                           |                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\$connString</code> | <p><b>SQL style:</b> <code>ENG=xx;UID=xx;PWD=xx</code></p> <p>where ENG, UID, PWD are the database engine name, database user ID, and password.</p> <p><b>ODBC style:</b> <code>DSN=xx</code></p> <p>where the engine corresponding to the ODBC data source <code>xx</code> is stopped.</p> |
| <code>\$timeout</code>    | Number of seconds the routine waits for the database to stop on each try. Default = 5 seconds.                                                                                                                                                                                              |
| <code>\$numRetries</code> | Number of times the routine attempts to stop the database before returning an error. Default = 10 retries.                                                                                                                                                                                  |

**unloadDbVersionData**

```
$ret = unloadDbVersionData ($dbh, $file);
```

Writes the contents of the DbVersion table into the specified file. Use this routine for backing up data and restoring utilities to match database versions.

**Input Arguments**

|                    |                  |
|--------------------|------------------|
| <code>\$dbh</code> | Database handle. |
|--------------------|------------------|

**Output Arguments**

|                     |                            |
|---------------------|----------------------------|
| <code>\$file</code> | Complete path to the file. |
|---------------------|----------------------------|

**Using the Database Utilities**

The following topics describe the available database utilities:

| Utility Name                   | Description                                                                                                                                                                                                   |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">backup.pl</a>      | Backs up the database file to a specified directory. It also backs up the files that are listed in manifest files into specific directory locations. For details, see <a href="#">backup.pl</a> , page 12-10. |
| <a href="#">configureDb.pl</a> | Performs several functions, including installing and uninstalling the database.                                                                                                                               |
| <a href="#">dbinit</a>         | Creates an empty database, assigns the default user ID and password, and specifies various options.                                                                                                           |
| <a href="#">dbMonitor</a>      | Ensures that a database can accept connections.                                                                                                                                                               |
| <a href="#">dbpasswd.pl</a>    | Changes the password field in the database configuration files.                                                                                                                                               |
| <a href="#">DBPing</a>         | Ensures that multiple database engines can accept connections at startup.                                                                                                                                     |

**CISCO CONFIDENTIAL**

| Utility Name                     | Description                                                                                                                            |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">dbreader.pl</a>      | A web-based Perl database query and manipulation utility intended to supplement the Sybase dbisqlc.                                    |
| <a href="#">dbRestoreOrig</a>    | The dbRestoreOrig utility restores a canned database from the orig directory.                                                          |
| <a href="#">dbvalid</a>          | A Sybase utility that determines if the database is valid.                                                                             |
| <a href="#">restorebackup.pl</a> | Restores a previous backup. For details, see <a href="#">restorebackup.pl</a> , page 12-11.                                            |
| <a href="#">runIsql</a>          | Runs the SqlAnywhere ISQL utility on the database identified by the connection string. The SQL commands are read from the script file. |

**configureDb.pl**

This utility has several functions:

```
perl configureDb.pl action={install|uninstall} <dsn=database>
```

If `action=install`, copies the database file from the /orig to the runtime directory. If `action=uninstall`, removes the database file.

```
perl configureDb.pl action=rebuild dsn=database
```

Rebuilds database files to 9.0.0 format. Can be called during upgrade from older versions.

```
perl configureDb.pl action={reg|unreg} dsn=database dmprefix=prefix
```

If `action=reg`, registers the database, including populating the .odbc.ini or Windows odbc registry, updating dmgttd.conf or the Windows services registry, and updating the DBServer.properties file. If `action=unreg`, unregisters the database.

```
perl configureDb.pl action=upgrade dsn=database portid=number
```

Checks the database version and upgrades to 9.0.0 format.

```
perl configureDb.pl action=upgradeall
```

Upgrades every database to 9.0.0 format.

```
perl configureDb.pl action=validate dsn=database
```

Use the dbvalid utility to validate the specified database.

```
perl configureDb.pl action=reg dsn=database dmprefix=prefix dbmonitor=no
```

Do not register [dbMonitor](#) as the default database monitor. This option is intended for use only when you want to substitute your own database monitor, such as [DBPing](#).

**CISCO CONFIDENTIAL****Input Arguments**

`dsn` The data source name (for example, cmf or rme)

`dmprefix` The system name (for example, CWCS or Essentials). For example, to register the Essentials database, enter:

```
perl configureDb.pl action=reg dsn=rme dmprefix=Essentials
```

The `configureDb.pl` script always validates `dmprefix` against the `dmprefix` value in `odbc.tmpl`. If the two do not match, the script will throw a warning and use the `odbc.tmpl` value. If there is no `dmprefix` entry in `odbc.tmpl`, the given entry is used and added to `odbc.tmpl`.

To find the value of `dmprefix`:

- On Solaris platforms, go to `$NMSROOT/objects/dmgt/dmgt.d.conf` and look for `{ $dmprefix }DbEngine`.
- On Windows platforms, use `regedit` to access `HKEY_LOCAL_MACHINE > SYSTEM > Current ControlSet > Services > { $dmprefix }DbEngine`.

`portid` The port ID number. This must be a 16-bit integer smaller than 65535.

**dbinit**

```
dbinit -j [-b] [-c] [-p page-size] dbName.db
```

**Note**


---

This procedure should be performed by developers only.

---

The `dbinit` utility is a Sybase utility that:

- Creates an empty database with a system catalog and system stored procedures.
- Assigns the following default user ID and password: DBA, SQL.
- Specifies the page size, transaction log file, case sensitivity and blank padding options.

**Prerequisites**

- CMF 1.2 or higher must be installed.

**Input Arguments**

`dbName` The name of the new database. The `.db` extension is required.

**Switches**

Use `dbinit -help` for additional help with switches.

`-j` Do not install runtime Java classes.

`-b` Pads blanks in strings for comparisons.



**CISCO CONFIDENTIAL**

-c Enforces case sensitivity for all string comparisons.

**Note** Case sensitivity cannot be changed later.

-p *page-size* Sets the page size.

**Note** Page size cannot be changed later.

**dbMonitor**

```
dbMonitor dsn -app daemon_registered_name -dbserver database_registered_server_name
[-sterror "start_error_interval"] [-stretry "start_retry_number"]
[-sleep "sleep_interval"] [-error "error_sleep_interval"]
[-retry "error_retry_number"] [-debug] [-nodisplay]
```

DbMonitor is the default database-engine process monitor. It ensures that a database can accept connections and avoid race conditions in database connections during engine startup, and periodically monitors the database to ensure that the connection is still available. Each database requires a separate DbMonitor process to monitor it. For example, all CWCS applications depend on the DbMonitor for the CWCS database; an application with its own database will require an additional DbMonitor instance for that database, and only that application will be dependent on that instance. Only the ODBC version of DbMonitor is used. See the “DBPing” section on page 11-56 for an alternative database monitor.

DbMonitor performs these functions:

- On startup, DbMonitor attempts to connect to the database and periodically selects from a common database table. If successful, it notifies the Daemon Manager to start the dependent applications.
- After the initial startup, DbMonitor periodically monitors the database by attempting selects from the common table. If it is unsuccessful, it sends a message to the front end and notifies the Daemon Manager to terminate the dependent applications.
- Important: All daemons dependent on the database should place a dependency on the corresponding DbMonitor and not on the database. This is because Daemon Manager does not accurately reflect the state of the database, but DbMonitor does.
- A DbMonitor entry is automatically created for each database that is registered using the [configureDb.pl](#) utility.

**Input Arguments**

*dsn* Data source name (for example, cmf).

**Switches**

-app Registered process name (for example, CmfdBMonitor).

-dbserver Registered database server name (for example, CmfdBEngine).

-sterror Number of seconds to sleep before next connection try. Optional.

-stretry Number of times to try to make a connection. Optional.

-sleep Number of seconds to sleep before checking the database engine. Optional.

**CISCO CONFIDENTIAL**

- error           Number of seconds to sleep before next fetch for a valid connection. Optional.
- retry           Number of times to try a fetch before declaring the connection is down. Optional.
- debug           Flag to send more debug information to log file. Optional.
- nodisplay       Flag to disable all log information. Optional.

**DBPing**

```
$NMSROOT/bin/cwjava com.cisco.nm.cmf.dbsevice2.DBPing -name daemon_registered_name -dsn database_server_name_list [-maxtry tries] [-timeout time] [-debug]
```

DBPing is an alternative to [dbMonitor](#). It ensures that all specified database engines are successfully initialized at startup. It *does not* poll the databases for connectivity thereafter. If all database engines are up and responding, the DBPing process notifies Daemon Manager that DBPing itself is up and running; otherwise, it will notify Daemon Manager that DBPing is down. All other daemons requiring database operations should register themselves as dependent on DBPing.

DBPing performs these functions:

- On startup, DBPing attempts to connect to each database in the -dsn list. If all of these attempts are successful, it notifies the Daemon Manager to start any dependent applications. If any of them are down, DBPing will generate an error
- Important: All daemons dependent on the databases in the -dsn list should place a dependency on DBPing.
- You can use DBPing only if you use the the [configureDb.pl](#) utility's `dbmonitor=no` option.

**Input Arguments**

- name           Registered DBPing process name (for example, DFMDbMonitor).
- dsn            A comma-delimited list of database server names (for example: DFMDbEngine, CmfDbEngine).

**Switches**

- maxtry         Number of times to attempt to make a connection with each of the database servers named in the -dsn list. Optional.
- timeout        Amount of time (in milliseconds) between attempts to make a connection with each of the database servers named in the -dsn list. Optional.
- debug         Flag to send more debug information to log file. Optional.

**CISCO CONFIDENTIAL****Remarks**

DBPing provides a monitor that can verify the status of multiple database engines at startup. It is intended for situations where your application has multiple databases, and you want to ensure that all database engines are fully initialized before Daemon Manager starts up any processes that depend on them. To use DBPing for this purpose:

1. Register each of your application databases using the `dbmonitor=no` option of `configureDb.pl`. This will prevent automatic creation of a DBMonitor instance for each of these databases. For example:
 

```
perl configureDb.pl action=reg dsn=MyAppDB1 dmprefix=prefix dbmonitor=no
perl configureDb.pl action=reg dsn=MyAppDB2 dmprefix=prefix dbmonitor=no
```
2. During your application installation, install and register with Daemon Manager a process that makes use of DBPing. Make sure you:
  - a. Assign your DBPing process a unique `-name` value. For example: `$NMSROOT/bin/cwjava com.cisco.nm.cmf.dbsevice2.DBPing -name MyAppDBMon -dsn MyAppDB1, MyAppDB2 -maxtry 10 -timeout 5000 -debug`. You must use this same name (for example, `MyAppDBMon`) when registering the DBPing process with Daemon Manager.
  - b. Use a `DmgrRegisterWR` (for C and C++) or `DmgrRegisterJavaWR` (for Java) call to register the DBPing process with Daemon Manager. Both registration calls allow you to specify a `-timeout` value for the DBPing process. For more information, see the “[DmgrRegister](#)” section on page 21-51.
  - c. Pass with your registration call a `-timeout` value that is less than the combined value of your DBPing process `-maxtry` and `-timeout` values. For example, if your DBPing process call specifies `-maxtry 10` and `-timeout 5000`, your `DmgrRegisterWR` or `DmgrRegisterJavaWR` `-timeout` value should be less than 50000.
3. Set all other processes that depend on the databases in the DBPing `-dsn` list with a dependency on your DBPing process.
4. Unregister the DBPing process if the application is uninstalled.

**dbpasswd.pl**

This utility changes the password field in the following database configuration files:

- The `odbc.tmpl` file.
- The ODBC registry (`.odbc.ini` for Solaris, or the registry for Windows).
- The database service property file (`DBServer.properties`) and, if specified, the private property file.

You can use this utility in a variety of ways, as shown below.

**Table 11-9** *dbpasswd.pl Usage Summary*

| Format                                                         | Action                                                                 |
|----------------------------------------------------------------|------------------------------------------------------------------------|
| <code>dbpasswd.pl all</code>                                   | Changes all database passwords.                                        |
| <code>dbpasswd cfile=configurefile</code>                      | Overrides database settings based on the configuration file specified. |
| <code>dbpasswd.pl listdsn</code>                               | Lists all available data sources in the product.                       |
| <code>dbpasswd.pl dsn=odbc_datasource</code>                   | Changes the database password.                                         |
| <code>dbpasswd.pl dsn=odbc_datasource npwd=new_password</code> | Changes the database password to <code>new_password</code> .           |

**CISCO CONFIDENTIAL****Table 11-9 dbpasswd.pl Usage Summary**

| Format                                                           | Action                                                                                         |
|------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| dbpasswd.pl dsn=odbc_datasource encryption=yes                   | Encrypts the database password.                                                                |
| dbpasswd.pl dsn=odbc_datasource encryption=yes npwd=new_password | Changes the database password to <code>new_password</code> and encrypts the database password. |

**Prerequisites**

- Daemon Manager must be shut down.
- On UNIX platforms, you must be logged in as root.
- On Windows platforms, you must be logged in as part of the local administrator group.
- This utility assumes that the databases and data sources are properly configured.

**Input Arguments**

|            |                                                                                                                                                                                                       |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| all        | Validates all registered databases and changes the password for each database.                                                                                                                        |
| cfile      | Overrides the database settings as specified in the configuration file mentioned                                                                                                                      |
| dsn        | Contains the ODBC data source name (DSN) of the database whose password will be changed. If the <i>dsn</i> argument is present, change the password for the specified database.                       |
| encryption | Encrypt the user name and password for the specified database. Possible values: YES, NO (default). The values are <i>not</i> case sensitive.                                                          |
| npwd       | Optional. If present, replace the password in the registry entry, <code>odbc.tmpl</code> , and the property file with <i>new_password</i> .                                                           |
| listdsn    | If present, this must be the only argument. Lists all data source names in the Solaris <code>.odbc.ini</code> or Windows registry and quits. To be included in this list, a database must be enabled. |

**Remarks**

Since `dbpasswd.pl` also tries the password in the `odbc.tmplorig` file if the value in the ODBC registry fails, it can be run as the last manual step in the existing database repair schemes that require copying the factory database from the *orig* directory.

The `dbpasswd.pl` utility validates passwords. Valid passwords must:

- Have a minimum of five and a maximum of 128 characters.
- Use alphanumeric characters (a-z, A-Z, 0-9) only. No special characters (e.g., #, \$, %) or spaces are allowed.
- Not have a number as the first character.

**Related Topics**

See the [“Updating the Database Password”](#) section on page 11-24.

**CISCO CONFIDENTIAL****dbreader.pl**

This is a web-based Perl database query and manipulation utility intended to supplement the Sybase dbisqlc.

**Note**


---

Dbreader cannot be invoked from the command line; it can only be run from a browser.

---

For information about using dbreader, see the “[Examining the Contents of a Database](#)” section on [page 11-31](#).

**Runtime Location**

`$NMSROOT/htdocs`

**dbRestoreOrig**

```
dbRestoreOrig.pl dsn=dsname dmprefix=dmprefixname [npwd=newpassword]
```

The dbRestoreOrig utility:

- Restores the pre-canned database from the orig directory.
- Changes the file permissions.
- Populates the encrypted or plain-text user ID and password to .odbc.ini or the registry entry.
- Populates the user ID and password to the property file for JDBC access (CWCS 2.2 and later).

**Note**


---

This utility was first introduced in CWCS 2.2. The JDBC property files were included in previous CWCS releases.

---

**Input Arguments**

`dsname` Data source name (for example, cmf).

`dmprefixname` Used to construct the database engine process name. For example:

- For CWCS: `dmprefix` is `Cmf` and the CWCS database engine name is `CmfDbEngine`.
- For RME: `dmprefix` is `Essentials`, and the RME database engine name is `EssentialsDbEngine`.

The `dmprefix` argument is initially configured using `configureDb.pl`. To determine the value of `dmprefixname`, see the “[configureDb.pl](#)” section on [page 11-53](#).

**Caution**


---

There is a check to validate this argument against the `dmPrefix` property in `odbc.tmpl`. If this property is absent, the user is prompted for action.

---

`newpassword` Optional. If present, replace the contents of the current password with *newpassword*.



**CISCO CONFIDENTIAL****Input Arguments**

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\$scriptFile</code> | Complete path name to the script file that contains the SQL commands.                                                                                                                                                                                                                                                                                                                                              |
| <code>\$connString</code> | <b>SQL style:</b> <code>ENG=xx;DBN=xx;UID=xx;PWD=xx</code><br>where ENG, DBN, UID, PWD are the database engine name, database name, database user ID and password. Connection string must be complete or this routine will fail.<br><b>ODBC style:</b> <code>DSN=xx</code><br>Contains either the DSN or the DSN plus the user ID and password. In either case, it is converted to SQL-style before the ISQL call. |

***CISCO CONFIDENTIAL***





CISCO CONFIDENTIAL

## CHAPTER 12

# Using Backup and Restore

---

This release of CWCS provides a database backup and restore framework based on the classical CMF backup and restore infrastructure. This framework is used by all CWCS-based applications, including RME, Campus, DFM, ~~DFM~~ and ACLM.

As of release 3.0, this framework can also restore the backup data from CMF 2.1 and CMF 2.2. Restoring a backup from a CMF release earlier than 2.1 is not supported.

CORE-based backup and restore is no longer supported.

The following topics describe using the CWCS backup and restore framework:

- [Using CWCS Backup](#)
- [Using CWCS Restore](#)
- [CWCS Backup and Restore API Command Reference](#)
- [Restoring a Corrupt Database](#)

For more information about CWCS backup and restore features, refer to *CMF2.3 Restore Framework Software Design Specification*, ENG-306868.

## Using CWCS Backup

The following topics describe using the CWCS backup framework:

- [CWCS Backup](#)
- [How CWCS Backups are Stored](#)
- [Running CWCS Backups](#)
- [Offline Backup](#)

## CWCS Backup

There is no change in the backup program for CWCS 3.0. However Campus Manager, ACLM, and DFM applications must change their application backup manifest directory structure as shown in [Table 12-1](#).

**CISCO CONFIDENTIAL****Table 12-1 Application Manifest Directory Changes for CWCS 3.0**

| <b>Application</b> | <b>Backup Directory</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Campus Manager     | <p>\$NMSROOT/backup/manifest/campus</p> <p><b>Note</b> Campus has its own restore adaptor:<br/>\$NMSROOT/bin/RestoreTools/campus/restore.pm.</p> <p><b>Note</b> If the backup is based on CMF 2.2, access the backup files under TEMP_FOLDER/cmf/campus, TEMP_FOLDER/cmf/ut and TEMP_FOLDER/cmf/ani.</p> <p><b>Note</b> If the backup is based on CWCS 3.0, access the backup files under TEMP_FOLDER/campus.</p>                                                                                                          |
| ACLM               | <p>\$NMSROOT/backup/manifest/aclm</p> <p><b>Note</b> ACLM has its own restore adaptor:<br/>\$NMSROOT/bin/RestoreTools/aclm/restore.pm.</p> <p><b>Note</b> If the backup is based on CMF2.2, access the backup files under TEMP_FOLDER/rme/aclm.</p> <p><b>Note</b> If the backup is based on CWCS 3.0, access the backup files under TEMP_FOLDER/aclm.</p>                                                                                                                                                                 |
| DFM                | <p>\$NMSROOT/backup/manifest/dfm</p> <p><b>Note</b> DFM has its own restore adaptor:<br/>\$NMSROOT/bin/RestoreTools/dfm/restore.pm</p> <p><b>Note</b> Irrespective of the version of CMF or CWCS, access the DFM data from TEMP_FOLDER/dfm.</p> <p>In CWCS 3.0, DFMfh is a module under DFM.</p> <p><b>Note</b> If the backup is based on CMF 2.1 or CMF 2.2, access the DFMfh data under TEMP_FOLDER/dfmfh.</p> <p><b>Note</b> If the backup is based on CWCS 3.0, access the DFMfh data under TEMP_FOLDER/dfm/dfmfh.</p> |

## How CWCS Backups are Stored

The CWCS backup and restore framework creates backup locations of the form  
*backupdir/generationnumber/suite/directory/filename*

Where:

- *backupdir* is the root directory where all backups are to be stored.
- *generationnumber* is the number of the backup. Directories are created in serial order, with the highest number representing the latest backup. For example: 1, 2, and 3, where 3 is the latest database backup.
- *suite* is the name of your application or suite. Often, this is the same as the data source (database) name. For example: For CWCS, the *suite* and database name are both “cmf”, but for Campus Manager, the *suite* is “campus” and the database name is “ani”.

## CISCO CONFIDENTIAL

- *directory* is the the database directory being backed up. Directories include the database directory and any suite application's data directories. It can also include the database template file, the CWCS (or CMF) version information, and filebackup.tar (which is the archive of all application configuration data files listed in the application backup manifest file datafiles.txt).
- *filename* is the name of the files that have been backed up. These files include database (.db), log (.log), the *dsn.txt* file, and database version file (*dsn\_DbVersion.txt*). Application directories will contain only the copy of datafiles.txt from the backup manifest-specified locations.

## Running CWCS Backups

To run a CWCS backup, each application must:

---

**Step 1** Create the backup manifest directory structure *\$NMSROOT/backup/manifest/suite* (see the “[Creating the Backup Manifest Files](#)” section on page 11-10).

**Step 2** Under that structure, create the */database/orig* subdirectory and place the *dsn.txt* file there. The *dsn.txt* file contains the information required to back up the suite database (see the “[Creating the Database Backup Manifest File](#)” section on page 11-11).

For example: For the CWCS suite, the database backup manifest file *cmf.txt* goes in *\$NMSROOT/backup/manifest/cmf/database/orig*.

**Step 3** Also under the *\$NMSROOT/backup/manifest/suite*, create the */app/orig* subdirectory for each application module to be backed up and place its matching *datafiles.txt* file there. Each *datafiles.txt* file contains the list of files to be backed up for that application module (see the “[Creating the Application Backup Manifest File](#)” section on page 11-11).

For example, the *configArchive* module of Resource Manager Essentials must place its *datafiles.txt* file under *\$NMSROOT/backup/manifest/rme/configArchive/orig*.

**Step 4** Configure the database backup manifest by running the following command:

```
$ENV{"NMSROOT"}/bin/perl $ENV{"NMSROOT"}/objects/db/conf/configureDb.pl action=install
dsn=dsn
```

Where *dsn* is your database name (e.g., for CWCS, *cmf*).

The script will copy the suite's *dsn.txt* file from the */orig* directory to its parent */database* directory, and replace *\$ENV{"NMSROOT"}* with the actual directory path. For example: For CWCS, the *cmf.txt* file will be copied from *\$NMSROOT/backup/manifest/cmf/database/orig* to *\$NMSROOT/backup/manifest/cmf/database*.

**Step 5** Configure the application backup manifest by running the following command:

```
$ENV{"NMSROOT"}/bin/perl $ENV{"NMSROOT"}/objects/db/conf/configureDb.pl action=install
app=app
```

Where *app* is your application module (e.g., for RME, *configArchive*).

The script will copy the application module's *datafiles.txt* file from its */orig* directory to its parent */app* directory, and replace *\$ENV{"NMSROOT"}* with the actual directory path. For example: For RME, the *datafiles.txt* file will be copied from *\$NMSROOT/backup/manifest/rme/configArchive/orig* to *\$NMSROOT/backup/manifest/rme/configArchive*.

**Step 6** The database and applications are now registered for CWCS backup and restore. To back them up, run the backup utility *backup.pl* (see the “[backup.pl](#)” section on page 12-10). The utility will find out what to back up from the database and application backup manifest files.

---

## CISCO CONFIDENTIAL

### Related Topics

See the “[configureDb.pl](#)” section on page 11-53.

## Offline Backup

Applications may require offline backup to avoid inconsistencies between the flat files and databases. If any of the application requires offline backup, then you can configure the backup mode as offline by specifying the value of `BACKUP_OFFLINE` parameter as `YES` in the `backup.properties` file under `$NMSROOT/conf/`. It should be done in respective applications install flow and should not be exposed to end-users.

If the offline backup is configured, then

- CiscoWorks Home page urgent message will be sent to all the users who are currently logged in before stopping daemon manager
- A JMS Event with the subject `cisco.mgmt.cw.cmf.backup` will be published before stopping daemon manager. Applications can listen for this event and can act accordingly.
- Daemons will be stopped during the backup process and will be started again after the backup is completed.

## Using CWCS Restore

The following topics describe using the CWCS restore framework:

- [CWCS Restore: Changes for CWCS 3.0](#)
- [Understanding the CWCS Restore Framework](#)
- [Running CWCS Restores](#)
- [Guidelines for Writing CWCS Restore Application Adaptors](#)
- [Sample CWCS Restore Application Adaptor](#)

## CWCS Restore: Changes for CWCS 3.0

For the CWCS 3.0 release:

- In previous releases, CMF applications placed their database files in the `<Application>/database` folder in the backup archive. Now, the restore framework:
  - Copies the files in the backup archive to the `TEMP_FOLDER/tempBackupData/<Application>` folder.
  - Extracts the `filebackup.tar` of each application to `TEMP_FOLDER/tempBackupData/<Application>/CSCOpX` (irrespective of the original `$NMSROOT` where it was backed up). This temporary directory structure is the same for Solaris and Windows.
- The Remote Upgrade tools are no longer available because restoring data from previous versions is incorporated into the restore programs. You can use the CWCS backup and restore programs to restore to CWCS 3.0 your data from older CMF 2.1 or 2.2.
- The Per-Product Restore feature is no longer available due to data inconsistencies caused by the replacement of the database during restore.

**CISCO CONFIDENTIAL****Understanding the CWCS Restore Framework**

In previous versions of the CWCS backup and restore framework, applications registered their files and folders to be backed up and restored in the file, `datafiles.txt`. The new framework extracts the backup data to a temporary directory, then calls the application's restore adaptors. Using the APIs and facilities provided by the framework, the adaptors handle the data conversion and store it in the runtime location.




---

**Note** New restore framework applications are required to write their own restore adaptors.

---

The new restore framework:

1. Extracts the backup data to a temporary location. For applications installed in the system that have data in the backup archive, it:
  - a. Copies the backup data of these applications to `TEMP_FOLDER/tempBackupData/<AppName>`
  - b. Extracts the `filebackup.tar` of the applications to `TEMP_FOLDER/tempBackupData/<AppName>/CSCOpX` folder. On Solaris platforms, the non-`$NMSROOT` files are extracted to the `TEMP_FOLDER/tempBackupData/<AppName>` folder.




---

**Note** The default location of the temporary directory is `$NMSROOT/tempBackupData`. You can use the `-t` option of the restore program to specify a different temporary directory.

---

2. For each of the following steps, the application restore adaptors are called in this pre-defined order: CMF/CWCS, Campus, RME, ACLM, and DFM.
  - a. Run `preRestore()` functions for all application adapters.
  - b. Run `doRestore()` functions for all application adapters.
  - c. Run `postRestore()` functions for all application adapters.




---

**Note** The application's restore adaptors are loaded from `$NMSROOT/bin/RestoreTools/<AppName>/restore.pm`.

---

3. Removes the temporary folder `TEMP_FOLDER/tempBackupData`.

The applications have knowledge of their data and the conversion logic. Therefore, the application's adaptor (`restore.pm`) will:

1. Extract the backup data from these locations:
  - `TEMP_FOLDER/tempBackupData/<AppName>/CSCOpX`
  - `TEMP_FOLDER/tempBackupData/<AppName>`




---

**Note** Applications access their archive data from this temporary directory. To find the location of this directory, application adaptors should use the `getTempFolder()` API.

---

2. Do the conversion if needed.

**CISCO CONFIDENTIAL**

3. Apply the converted file to the running machine.

## Running CWCS Restores

To run a CWCS restore operation, each application must:

- 
- Step 1** Write a Perl adaptor that takes care of applying application data to the runtime structure (see the [“Guidelines for Writing CWCS Restore Application Adaptors”](#) section on page 12-6).
- Step 2** Register the application with CWCS restore by placing the restore adaptor (the Perl module) in `$NMSROOT/bin/RestoreTools/<Application Name>`
- Step 3** Run the restore utility (see the [“restorebackup.pl”](#) section on page 12-11).
- 

## Guidelines for Writing CWCS Restore Application Adaptors

For CWCS 3.0, the new restore framework requires that all applications using the framework write their own application adaptors. When you write an application adaptor, follow these guidelines:

- All application adaptors must be implemented as Perl modules. To implement a restore program in Java or any other language, supply a Perl module to call the non-Perl application.
- All application adaptors must be named `restore.pm`.
- Use the APIs provided by the CWCS backup and restore framework (see the [“CWCS Backup and Restore API Command Reference”](#) section on page 12-9).
- Place the `restore.pm` files in subdirectories of the `RestoreTools` folder in the `bin` directory. They will be identified based on the directory in which they are placed.

`$NMSROOT/bin/RestoreTools/<Application>/restore.pm.`

For example, RME’s application adaptor is stored in:

`$NMSROOT/bin/RestoreTools/rme/restore.pm`

- All application restore modules must contain four functions. These functions are called by the restore framework.
  - `preRestore()`. The `preRestore()` function should not make any changes in the running machine that breaks the applications. Follow this principle: if even one application adapter’s `preRestore()` fails, restore the system to the state it was in before the restore program was run.




---

**Note** This function can be empty but must be declared.

---

- `doRestore()`. Applications should use only the `doRestore()` function to apply data to the machine.
- `postRestore()`. The `postRestore()` function cannot terminate the restore. This function is used to do fine tuning after the restore.




---

**Note** This function can be empty but must be declared.

---

**CISCO CONFIDENTIAL**

- UNLOAD\_restore(). The UNLOAD\_restore() function removes the functions defined in this restore.pm from memory.



**Note** If applications are using any other functions in their restore.pm, they should add an entry for those functions to this list.

- The last line of the restore.pm must be "1;" to indicate the end of the module.
- When writing the application adapter, use the format shown in [Example 12-1](#).

**Example 12-1 CWCS Restore Application Adapter Format**

```

sub preRestore
{
 # Code for pre restore of this application;
 return 0; # But on error should return non-zero.
 # Returning non-zero will stop the restore.
}
sub doRestore
{
 # Code for the actual restore.
 return 0; # But on error should return non-zero.
 # Returning non-zero will stop the restore.
}
sub postRestore
{
 # Code for post restore operation
}
sub UNLOAD_restore
{
 # This function removes the functions defined in this restore.pm from memory.
 # The following three entries are required. If applications are using any
 # other functions in their restore.pm, they should add an entry for
 # those functions here.
 undef &preRestore;
 undef &doRestore;
 undef &postRestore;
}
1;

```

- Before calling the application's module, the restore framework redirects the STDOUT and STDERR to the restorebackup.log file. Therefore, applications can print their messages to STDOUT using print statements, which will be captured in restorebackup.log. This is the default behavior. However, this means that none of an application's messages will be displayed in the CLI. To add user input to a restore adapter:
  - Before calling user input functions, call the redirectToScreen () API (see the [“redirectToScreen” section on page 12-16](#)). This redirects further outputs of STDOUT to print on the screen.
  - After calling the user input functions, call the redirectToLog () API (see the [“redirectToLog” section on page 12-17](#)). This will redirect further outputs of STDOUT back to the restore log file.
- When reporting errors, follow these guidelines:

**CISCO CONFIDENTIAL**

- The modules should not use Perl's exit() function to exit the program. They should return to their respective methods with the appropriate return values.
- The preRestore() functions should return zero for success and non-zero for any error.
- The doRestore() functions should return zero for success and non-zero for any error.
- The return status of the postRestore() function is ignored by the framework.

**Related Topics**

See the:

- [“Sample CWCS Restore Application Adaptor” section on page 12-8.](#)
- [“CWCS Backup and Restore API Command Reference” section on page 12-9.](#)

## Sample CWCS Restore Application Adaptor

**Example 12-2** assumes that the backup archive for CMF2.1, CMF2.2, and CWCS 3.0 contains the following files for this application:

- CMF2.1
  - d:\program Files\CSCOpX\conf\file1.conf
  - d:\Program Files\CSCOpX\etc\file2.data
- CMF2.2
  - d:\CW2000\conf\file1.conf
  - d:\CW2000\etc\file2.data
  - d:\CW2000\etc\file3.data
- CWCS 3.0
  - d:\CW2000\conf\file1.conf
  - d:\CW2000\etc\file2.data
  - d:\CW2000\etc\file3.data

In this example:

- file1.conf is not changed between the three versions.
- file2.data of CMF2.1 needed conversion when restored in CWCS 3.0.
- file3.data was introduced in CMF2.2 and does not need any conversion in CWCS 3.0.

**Example 12-2 Sample restore.pm File**


---

```
sub preRestore {return 0;}
sub doRestore
{
 if (getCMFVersion() eq "2.1")
 {
 if (CopyFileToNMSROOT("conf\file1.conf")!=0) {return -1;}
 if (convertData2()!=0) {return -1;}
 }
 elsif (getCMFVersion() eq "2.2")
 {
```



**CISCO CONFIDENTIAL**

```

 if (CopyFileToNMSROOT("conf\file1.conf")!=0) {return -1;}
 if (CopyFileToNMSROOT("etc\file2.data")!=0) {return -1;}
 if (CopyFileToNMSROOT("etc\file3.data")!=0) {return -1;}
}
elseif (getCMFVersion() eq "3.0")
{
 if (CopyFileToNMSROOT("conf\file1.conf")!=0) {return -1;}
 if (CopyFileToNMSROOT("etc\file2.data")!=0) {return -1;}
 if (CopyFileToNMSROOT("etc\file3.data")!=0) {return -1;}
}
return 0;
}
sub convertData2
{
my $NMSROOT, $d, $APP, $TEMP_FOLDER;
$NMSROOT= getNMSROOT(); $d= getFolderSeperator();
$TEMP_FOLDER = getTempFolder(); $APP="cmf";
 SourceFile=TEMP_FOLDER+$d+$APP+$d+"CSCOpX"+"$d"+"etc"+d$+"file2.dat;
 DestinationFile=$NMSROOT+$d+"CSCOpX"+"$d"+"etc"+d$+"file2.dat;
 SourceHandler = OpenFile(SourceFile,<Read Mode>);
 DestinationHandler = OpenFile(DestinationFile, <Write Mode>);
Read content of SourceHandler, convert, write to DestinationHandler
close SourceFileHandler, DestinationFileHandler
Return 0 for success, -1 for any errors.
}

sub postRestore { }
sub UNLOAD_restore
{
 undef &preRestore; undef &doRestore; undef &convertData2;
 undef &postRestore;
}
1;

```

## CWCS Backup and Restore API Command Reference

The following topics describe the utilities and APIs provided by the CWCS framework.

Use these utilities to backup and restore your databases:

- [backup.pl](#)
- [restorebackup.pl](#)

Use these APIs when you write CWCS restore adaptors for your applications:

- [copyFileToNMSROOT](#)
- [copyFolderToNMSROOT](#)
- [getCMFVersion](#)
- [getCMFPatchVersion](#)
- [getNMSROOT](#)
- [getArchiveNMSROOT](#)
- [getFolderSeperator](#)
- [getLogFileName](#)
- [getTempFolder](#)

**CISCO CONFIDENTIAL**

- [isWindows](#)
- [redirectToLog](#)
- [redirectToScreen](#)
- [restoreDatabase](#)
- [StandardDbRebuild](#)

**backup.pl**

```
$NMSROOT/bin/perl $NMSROOT/bin/backup.pl backdir logfile numberGen
```

Backs up the database file to a specified directory. It also backs up the files that are listed in the backup manifest files into specific directory locations.

The backup script requires the following information:

- The location of the databases  
The database backup manifest file contains a list of database file names. The backup.pl script backs up all specified database files.
- A list of directories containing data files  
The application backup manifest file contains a list of directory names. The backup.pl script backs up all files in the specified directories.

**Input Arguments**

|                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>backdir</code>   | Backup (target) directory. Must be writable. The full directory path is required. The directory will be created if it does not exist.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <code>logfile</code>   | The name of the backup log file. When run from the command line: If specified, the full path is required and the parent directory must exist and be write-enabled (if not, it will <i>not</i> create the directory and will throw the message <code>Error: Cannot save STDERR to the log file</code> ). If not specified, the log will be written to <code>STDOUT</code> . From the GUI: The log is written to <code>\$NMSROOT/log/dbbackup.log</code> ; there is no option to specify a log file.                                                                         |
| <code>numberGen</code> | Number of backup archives to retain in the directory. Must be an integer value or blank. If blank, it creates or overwrites archive 0 and keeps all other archives. If specified, it creates a new archive and keeps only that number of archives. For example: You have 100 backups; archive 1 is the oldest and 100 is the newest. If <code>numberGen</code> is 5, backup.pl creates a new archive 101, deletes archives 1-96, and keeps archives 97-101. On the next run with the same <code>numberGen</code> value, it creates new archive 102 and deletes archive 97. |

**Remarks**

- You must specify the full path to both perl and the backup.pl script. You must also use the version of Perl supplied with CWCS.
- You can run this script from the CWCS desktop by selecting **Server Configuration > Admin > Backup**. This option lets you run a backup immediately or schedule it for a later date and time.

**CISCO CONFIDENTIAL**

- If a backup fails due to a previous backup being aborted or interrupted, you will see the following error message

".../backup.LOCK file exists. Most probably another backup process is running"

If you are sure that no another backup process is running, you can delete the backup.LOCK file. The purpose of this file is to prevent more than one instance of the backup process.

**Related Topics**

See the [“Creating the Backup Manifest Files”](#) section on page 11-10.

**restorebackup.pl**

```
perl restorebackup.pl -d backupdir [-t tempdir] [-gen generation]
```

Restores a previous backup. This script is run on demand from the command line with root privileges. The script will verify that no applications are running when it is invoked.

**Note**


---

Shut down the Daemon Manager before starting this utility.

---

**Runtime Location**

*\$NMSROOT/bin*

**Switches**

|                        |                                                                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>-d backupdir</i>    | Path to the backup directory. Required.                                                                                                                                                                                      |
| <i>-gen generation</i> | Generation to be restored. By default, restores the latest generation. Optional.                                                                                                                                             |
| <i>-t tempdir</i>      | Specifies a different temporary directory. Optional.<br>The restore framework uses a temporary directory to extract the contents of the backup archive. By default the temporary directory is <i>NMROOT/tempBackupData</i> . |

For example, to restore the fourth backup of Campus Manager from the MyDir directory, enter:

```
perl restorebackup.pl -d /MyDir -gen 4
```

**Remarks**

- This script restores the database files to the location specified in the appropriate *{dsn}.txt* file in the manifest directory for the database. This ensures that the database is restored to the proper location if the database is moved after a backup.
- This script uses the tar command to untar *filebackup.tar* for the data files indicated in the *datafiles.txt* file for the application. The *datafiles.txt* file is located in the following directory path:

*\$NMSROOT/backup/manifest/<Application Name>/<module Name>*

**CISCO CONFIDENTIAL**

- For Windows platforms, the `untar` command gets a “Permission denied” error if the original backup directory and its files do not have read permission to the current user. Therefore, application data files must be readable.

**Related Topics**

See the [“Guidelines for Writing CWCS Restore Application Adaptors”](#) section on page 12-6.

**copyFileToNMSROOT**

Copies the specified file from the backup archive to `$NMSROOT`.




---

**Note** Call this function only when there are no data conversions for the file.

---

**Syntax**

```
copyFileToNMSROOT (String <Application>, String <FileName>)
```

**Input Arguments**

|             |                                                                                                                                                                                                                                                                                         |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Application | The name of the application suite (for example, “RME”).                                                                                                                                                                                                                                 |
| FileName    | File to be copied from the backup archive.<br>Do not include <code>\$NMSROOT</code> . For example, if the file to be restored is:<br><code>d:\program files\CSCOpX\cmf\objects\web\conf\ssl.conf</code><br><br><FileName> should contain:<br><code>cmf\objects\web\conf\ssl.conf</code> |

**Return Values**

|          |         |
|----------|---------|
| 0        | Success |
| non-zero | Failure |

**copyFolderToNMSROOT**

Copies the complete folder from the backup archive to NMSROOT.

**Syntax**

```
copyFolderToNMSROOT (String <Application>, String <Folder>)
```

**CISCO CONFIDENTIAL****Input Arguments**

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| Application | The name of the application suite (for example, “RME”).                   |
| Folder      | Folder to be copied from the backup archive.<br>Do not include \$NMSROOT. |

**Return Values**

|          |         |
|----------|---------|
| 0        | Success |
| non-zero | Failure |

**getCMFVersion**

Returns the CMF or CWCS version of the backup data.

**Syntax**

```
getCMFVersion()
```

**Input Arguments**

None

**Return Values**

|             |                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------|
| CMF version | The CMF version of the backup data. Possible values:<br>“2.1” for CMF2.1<br>“2.2” for CMF2.2<br>“3.0” for CWCS 3.0 |
|-------------|--------------------------------------------------------------------------------------------------------------------|

**getCMFPatchVersion**

Returns the PATCHVER of the CWCS.



**Note** This API is valid for CWCS 3.0 only; it cannot detect the patch version for CMF 2.1 or CMF 2.2.

**Syntax**

```
getCMFPatchVersion()
```

**Input Arguments**

None

**CISCO CONFIDENTIAL****Return Values**

PATCHVER                    The patch version of CWCS. Possible values are “1”, “2”, and so on.

**getNMSROOT**

Returns the current \$NMSROOT.

**Syntax**

```
getNMSROOT()
```

**Input Arguments**

None

**Return Value**

path                        The current \$NMSROOT (the path where CiscoWorks is installed).

**getArchiveNMSROOT**

Returns the \$NMSROOT of the backup data.

**Syntax**

```
getArchiveNMSROOT()
```

**Input Arguments**

None

**Return Value**

path                        The \$NMSROOT of the backup data (relative to the path where CiscoWorks is installed).

**getFolderSeperator**

Returns the folder separator depending on the OS in which CWCS 3.0 is running.

**Syntax**

```
getFolderSeperator()
```

**Input Arguments**

None

## CISCO CONFIDENTIAL

### Return Values

separator                      Folder separator. Possible values:  
"/" for Solaris  
"\" for Windows

## getLogFileName

Returns the full path of the restorebackup.log.

### Syntax

```
getLogFileName()
```

### Input Arguments

None

### Return Values

filename                      The location of the log file, restorebackup.log.

## getTempFolder

Returns the temporary directory.

### Syntax

```
getTempFolder()
```

### Input Arguments

None

### Return Values

folder name                      By default, the temporary directory for restore is  
\$NMSROOT/tempBackupData.  
  
However, if the temporary path has been customized (using the -t parameter),  
the user-specified temporary path is returned.

## isWindows

Returns the OS type.

### Syntax

```
isWindows()
```

**CISCO CONFIDENTIAL****Input Arguments**

None

**Return Values**

OS type                      The operating system type. Possible values are “0” for Windows, “1” for Solaris

**redirectToScreen**

Reverses the redirectToLog() API, restoring STDOUT and STDERR to their default values. Reverses the redirectToLog() API,




---

**Note**      Applications should not use this API unless they need user inputs.

---

**Syntax**

```
redirectToScreen()
```

**Input Arguments**

None

**Return Values**

0                              Success

non-zero                      Failure

**Example**

By default, all messages produced by the applications are redirected to restorebackup.log. However, if an application adaptor needs to display some information on the screen to get input from a user, it should use the redirectToScreen() and redirectToLog() APIs. For example:

```
sub doRestore()
{
 ...
 print "<some print message>; # this will be redirected to the log
 redirectToScreen();
 print "<Print the message which should be displayed on the screen,
 which is used for confirmation by users (y/n) >";
 # the above print message will be printed on the screen.
 redirectToLog();
 # Any subsequent print statements will be redirected to the log only
 print "<other print messages>"; # will be redirected to the log
}
```

**Related Topics**

See the [“redirectToLog” section on page 12-17](#).



**CISCO CONFIDENTIAL**

## redirectToLog

Directs STDOUT and STDERR to the restore log file. Subsequent outputs of STDOUT and STDERR are captured in the log file.



---

**Note** Applications should not use this API unless they need user inputs.

---

**Syntax**

```
redirectToLog()
```

**Input Arguments**

None

**Return Values**

|          |         |
|----------|---------|
| 0        | Success |
| non-zero | Failure |

**Related Topics**

See the [“redirectToScreen”](#) section on page 12-16.

## restoreDatabase

The restoreDatabase API performs the following operations:

- Copies the backup database to the current database.
- Uses the password of the database being restored and updates all files accordingly.
- If the backup database password is not encrypted, it is encrypted and stored during the restore process (provided the current database configuration supports database password encryption).
- If the restore is across CMF versions (and therefore across application versions), the database is rebuilt to the current Sybase file format.

Any other operations must be handled in the application wrapper.

**Syntax**

```
restoreDatabase(<dsn>, <dmprefix>, [<suite>])
```

**Input Arguments**

|     |                            |
|-----|----------------------------|
| dsn | Database data source name. |
|-----|----------------------------|

**CISCO CONFIDENTIAL**

|          |                                                                                                                                                                                                                                                                                                                                                                        |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dmprefix | The daemon manager prefix. This is the prefix that is used to register the database with the Daemon Manager.                                                                                                                                                                                                                                                           |
| suite    | Application suite as registered with the backup framework.<br>If the dsn and the suite are different, then the suite name must be provided. If the suite name is not provided, the dsn is assumed to be the suite name.<br>The suite name and dsn are usually the same, with some exceptions such as Campus Manager, where the suite is “campus” and the dsn is “ani”. |

**Return Values**

|   |         |
|---|---------|
| 0 | Success |
| 1 | Error   |

**StandardDbRebuild**

Upgrades the database file format for a given suite or database data source name. This API unloads the data, creates a new database file, and loads the data back into this file.

This API is called internally in `restoreDatabase()` API when a restore across versions is detected. Applications can use it as necessary, but are not required to call it separately during a restore operation. For more information, see the [“configureDb.pl” section on page 11-53](#).

**Syntax**

```
InstallUtility::StandardDbRebuild(<suite>)
```

**Input Arguments**

|       |                                             |
|-------|---------------------------------------------|
| suite | Database data source name or the suite name |
|-------|---------------------------------------------|

**Return Values**

None.

**Restoring a Corrupt Database**

If the database has been corrupted, choose one of these solutions:

- Option 1: [Restoring a Corrupt Database from a Previous Backup](#)
- Option 2: [Recovering Part of a Corrupt Database](#)
- Option 3: [Abandoning a Corrupt Database](#)

**CISCO CONFIDENTIAL**

## Restoring a Corrupt Database from a Previous Backup

Use the backup and restore utilities for regular database maintenance.



**Note** Be sure to back up your database regularly.

### Related Topics

- See the “[backup.pl](#)” section on page 12-10.
- See the “[restorebackup.pl](#)” section on page 12-11.

## Recovering Part of a Corrupt Database

If only part of a database is corrupted, perform the following steps to save your data. Use the procedure appropriate to the platform:

### Recovering Part of a Corrupt Database On Windows Platforms

- Step 1** Log in as local administrator.
- Step 2** To stop the Daemon Manager, enter:
- ```
net stop crmdmgt
```
- Step 3** Copy the database files, including any log files, to a save directory. Always back up the original data.
- Step 4** If the dbvalid utility reports that one or more tables are corrupted, try replacing just those tables.
- For example, assume that the Essentials (RME) syslog tables SLG_MSG_UMGD and SLG_MSG are corrupted, which means that the data in these tables is gone. All you can do is remove the bad tables and replace them with the copies in the /orig directory. Then rerun dbvalid:
- ```
rm $NMSROOT/objects/db/syslog.db
copy $NMSROOT/objects/db/orig/syslog.dborig $NMSROOT/objects/db/.
dbeng8 -f rme.db
dbvalid -c "uid=dba;pwd=c2kY2k;dbf=rme.db"
```
- where *\$NMSROOT* is the directory in which the product was installed.
- Step 5** If replacing the corrupted tables doesn't work, try extracting the data from the database and reloading it into a new database. Using the same example from Step 4:
- ```
cd $NMSROOT/databases/rme
dbunload -c "uid=dba;pwd=c2kY2k;dbf=rme.db" savedir
```
- Notice the file reload.sql under the current directory and the *.data files under savedir.
- Now create the rme.db and syslog.db files, then rerun dbvalid:
- ```
rm -f rme.db, syslog.db, rme.log
dbinit -p 4096 rme.db
dbisqlc -c "uid=dba;pwd=sql;dbf=rme.db" -q read reload.sql
```
- The reload utility reloads the database and restores the old user ID and password values.

**CISCO CONFIDENTIAL**

**Note** The reload process might take a long time. Depending on the size of the database, reloading may run for several hours.

Then rerun dbvalid:

```
dbvalid -c "uid=dba;pwd=c2kY2k;dbf=rme.db"
```

**Step 6** To restart the Daemon Manager, enter:

```
net start crmdmgt
```

**Step 7** Optional: Use dbreader to examine the contents of the database (see the [“Examining the Contents of a Database”](#) section on page 11-31).

## Recovering Part of a Corrupt Database On Solaris Platforms

**Step 1** Log in as root.

**Step 2** To stop the Daemon Manager, enter:

```
/etc/init.d/dmgt stop
```

**Step 3** Copy the database files, including any log files, to a save directory. Always back up the original data.

**Step 4** Set the environment variables (see the [“Setting Up Your Environment”](#) section on page 11-19).

**Step 5** Create a directory to keep the data:

```
mkdir savedir
dbunload -c "uid=$uid;pwd=$pwd;dbf=$dbfile.db" savedir
```

where *\$uid* and *\$pwd* are the user ID and password for your database.

Notice the file reload.sql under the current directory and the \*.data files under savedir.

**Step 6** To create the \$dbfile.db file, enter:

```
rm -f $dbfile.db $dbfile.log
```

**Step 7** To initialize the \$dbfile.db file, enter:

```
dbinit -p 4096 $dbfile.db
dbisqlc -c "uid=$uid;pwd=$pwd;dbf=$dbfile.db" -q read reload.sql
```

where *\$uid* and *\$pwd* are the default user ID and password values (dba and sql).

The reload utility reloads the database and restores the old user ID and password values (used in Step 4).

**Step 8** Optional: Use dbvalid to determine if the database is valid (see the [“dbvalid”](#) section on page 11-60).

**Step 9** To restart the Daemon Manager, enter:

```
/etc/init.d/dmgt start
```

**Step 10** Optional: Use dbreader to examine the contents of the database (see the [“Examining the Contents of a Database”](#) section on page 11-31).

**CISCO CONFIDENTIAL**

## Abandoning a Corrupt Database

If the data in the database is not important, or the database is totally corrupted, you can copy a clean database from the orig directory. Assuming that the database file is the only problem, this, at least, will allow you to avoid a reinstallation.

To reinitialize the database, use the dbRestoreOrig.pl script (see the [“dbRestoreOrig”](#) section on page 11-59).

***CISCO CONFIDENTIAL***



CISCO CONFIDENTIAL

## CHAPTER 13

# Using the Core Client Registry

---

The Core Client Registry (CCR) manages the seamless installation, upgrade, patching and uninstall of Multiple Device Contoller (MDC) modules and the Core module itself.

The following topics describe how to use CCR to manage these tasks:

- [Understanding CCR, page 13-23](#)
- [About the CCR Components, page 13-24](#)
- [About CCR System Flow, page 13-27](#)
- [About CCR Data Structures, page 13-28](#)
- [Using the CCR C++ API, page 13-31](#)
- [Using the CCR API: Example, page 13-46](#)
- [Using the CCRAccess Client, page 13-47](#)
- [Scripting CCRAccess, page 13-49](#)
- [Using the CCRAccess DLL, page 13-50](#)
- [Using the CCR Java Interface, page 13-51](#)
- [Encrypting and Decrypting CCREntry Values, page 13-52](#)

For more information about CCR, refer to the *Core Client Registry Software Unit Design Specification*, EDCS ENG-129945.

## Understanding CCR

The Core Client Registry:

- Tracks overlaps in module requirements to prevent addition of modules that already exist.
- Maintains these requirements when modules are removed that are still needed for execution of other modules.
- Stores information needed to instantiate and run a module properly.

CCR tracks:

- Application locations.
- Application configurations.
- MDC client extension libraries.
- Namespaces.

**CISCO CONFIDENTIAL**

- Initialization information.
- Registry and environmental entries.

CCR can:

- Add entries.
- Update configuration files.
- Remove entries.
- Track entry references.
- Increment references.
- Decrement the references.

You can access CCR from both C++ and Java applications, usually in the form of a servlet. CCR uses:

- XML to store the data.
- Generic C++ libraries and STL to allow porting to other operating systems.
- JNI to create a bridge from C++ to Java.
- Xerces to manage the DOM tree.

**Note**


---

CCR is an information container only. It does not define the meaning of the content; this is up to the component developer. For example, an application can register user roles and the Core Admin Module (CAM) will pick them up. CCR will not understand the format and information needed for this kind of entry. The contract is between CAM and applications.

---

## About the CCR Components

CCR has five major components, which are described in the following topics:

1. [CCR Local System Data \(LSD\) Component](#)
2. [CCRProcess Component](#)
3. [CCRInterface Component](#)
4. [CCREntry Component](#)
5. [CCRResponse Component](#)

### CCR Local System Data (LSD) Component

LSD is the data structure that contains the registration information. It is the repository for all the entries tracked by the CCR. It is in the form of an XML file and is loaded by the CCRprocess. The XML file contains one root element called CCRRoot. CCRRoot will always have an element called *resources* that contains all of the resources that have been added to an MDC. All of the other elements under CCRRoot are MDCs.



**CISCO CONFIDENTIAL****Resources**

If any resources are added to CCR, then CCRRoot will have a Resources element. Elements within Resources must adhere to specific rules. Each element is followed by a number that allows differentiation of the resources.

All immediate children of Resources will be called either Library, Custom, ApacheConf, or InitializationInfo followed by a number. For example, Custom1, Library2, ApacheConf5, InitializationInfo7.

These elements are:

- Library—describes any kind of library.
- InitializationInfo—describes any initialization information for the MDCs.
- ApacheConf—describes a change to an Apache configuration file.
- Custom—describes any other kind of resources.

All other elements under CCRRoot are assumed to be MDC elements:

- ExtensionLibraries.
- Configurations.
- Java.
- Notifications (this is deprecated).
- Logging.

All these elements (except for Logging) will contain children that conform to the Resources child element rules (Custom, Library, etc.).

All of these four types of child elements can have the following types of children.

- `<name>value</name>`
- `<data>value</data>`
- `<location>value</location>`
- `<custom_name>value</custom_name>`

These can be changed by privileged users.

The Logging child will have only one of the following elements:

```
<Location>log_file_name_w_location</Location>
```

However, it can have any number of these elements:

```
<categoryname priority="priority value"/>
```

Where the priority value can be DEBUG, INFO, WARN, ERROR, FATAL.

**CISCO CONFIDENTIAL****CCRProcess Component**

This is the heart of CCR. It includes the code that loads the XML file as well as the code that manipulates the elements.

This is the thread that waits for requests to act on the LSD. Upon instantiation, the CCRProcess will either load up the existing LSD into a DOM tree, or if one is not available, it will create a new skeleton LSD that will become the new default one. It also makes backups of the LSD at certain intervals to prevent information from being lost or corrupted. It also creates backups when the LSD has been updated to allow recovery of previous versions if necessary.

There should only be one instance of CCRProcess per process. It uses the sync libraries to prevent deadlocking when multiple CCRInterfaces access the CCRProcess.

**CCRInterface Component**

The CCRInterface allows modules to modify the data based on their needs (like installation, removal). It is the access point for all CCR functionality. The CCRInterface starts the CCRProcess (if it has not been started already) and it also communicates requests to it. JNI is used to provide the servlets with a means to access the daemon from Java.

C++ client uses a CCRInterface object to interact with the CCR.

Two objects are primarily used with CCRInterface: CCREntry and CCRResponse.

**CCREntry Component**

Most CCRInterface functions will involve the addition, subtraction, or manipulation of CCREntry objects. These basically contain a list of `std::string` objects that will allow the CCRProcess to find entries or to properly place the entries with the LSD. It is important to use `std::string` objects as this will provide for easy translation between Java and C++. CCREntry also provides encryption & decryption capabilities

There are three important fields that help CCR find a CCREntry object within the DOM tree.

- `rootElement`—This field describes either the name of the MC, or it is *resources*. It has getter/setter methods.
- `subElement`—This field describes the child element of the `rootElement`. If the root is an MC it could be Libraries, etc. If it was Resources then it could be Custom1, etc.
- `type`—This field is one of the allowable child types for Resources.

**CCRResponse Component**

This contains the response information for many of the CCRInterface retrieval function calls. It has a Java corollary.

It contains a success value and a vector full of CCREntry objects that were retrieved.

## CISCO CONFIDENTIAL

# About CCR System Flow

This topic provides an overview of the major functions of CCR:

- [Adding an LSD Entry \(Installation\)](#)
- [Removing an LSD Entry \(Uninstall\)](#)
- [Modifying an LSD Entry \(Patching/Upgrading\)](#)
- [Retrieving an LSD Entry](#)

## Adding an LSD Entry (Installation)

The requesting process determines whether or not the CCRProcess has been started.

It creates an CCREntry with the following fields:

- Rootcomponent: The tree within the LSD where the new entry should reside.
- Subdirectory: The subdirectory where the entry will reside within the rootcomponent.
- Resourcetype: The type of resource that is being stored.
- Resourcedata: An actual string representation of the resource.

The CCREntry is passed to the CCRInterface method that handles the addition of entries.

The existing references are searched to determine whether the entry already exists. If it does, it is added, and the resource reference count is incremented. If it does not exist, it is added, and the resource is also added with the new reference and reference count of 1.

## Removing an LSD Entry (Uninstall)

The requesting process determines whether or not the CCRProcess has been started.

It creates an CCREntry with the following fields:

- Rootcomponent: The tree within the LSD where the entry resides.
- Subdirectory: The subdirectory where the entry resides within the rootcomponent.
- resourcetype: The type of resource that is being removed.

The CCREntry is passed to the CCRInterface method that handles the removal of entries. If the resource is referenced by only one rootcomponent, then it is completely removed. Otherwise the resource's reference count is decremented by one and the actual reference of the component below the resource is removed.

## Modifying an LSD Entry (Patching/Upgrading)

The requesting process determines whether or not the CCRProcess has been started. The requesting process then locates the entry to be updated. It either creates or is given the entry information.

The entry is updated. First the current entry is removed. If the old entry was the last reference to the resource, that resource is removed.

**CISCO CONFIDENTIAL**

The existing references are searched to determine whether the entry already exists. If it does, it is added, and the resource reference count is incremented. If it does not exist, it is added, and the resource is also added with the new reference and reference count of 1.

## Retrieving an LSD Entry

Pertinent information in the retrieval of the entry is entered into an `CCREntry`.

The retrieval request is submitted. If the information in the entry correlates to more than one entry in the repository, several entries will be returned in a `std::list` object. Otherwise the `std::list` object will only contain one entry. If no entry is found, the `std::list` object will be empty.

## About CCR Data Structures

This topic describes the data structures for the following:

- [Local System Data \(LSD\) Data Structure](#)
- [CCREntry Data Structure](#)
- [CCRResponse Data Structure](#)

## Local System Data (LSD) Data Structure

The LSD is an XML document that will contain all of the relevant registration information. There will be a root element which will contain:

- Core registration information
- MDC registration information
- Resources element

All of the leafs of the Core and MDCs will refer to elements within the resources subtree. These include location data, Apache configuration data, library data, initialization data, Java libraries, logging information, notification information, and any custom data that an MDC might need. The resources subtree will maintain the data, location, reference count, and the specific reference of each resource.

Whenever a new resource is to be added to the Core or an MDC, the resource subtree should be checked to see if there is a duplicate resource already available. If the resource is already available, the reference in the Core or MDC branch should point to the existing resource, the reference count of the resource should be incremented and the MDC that is referencing the resource should be added as a child of the resource. Otherwise, a new resource is added and the MDC or Core will point to that.

```
- <CCRRoot>
 - <Resources>
 - <Library1>
 <Name>ite-nosd.dll</Name>
 <Location>c:\ismg\core\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <Core />
 </References>
 </Library1>
 - <Library2>
 <Name>rulesd.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
```

**CISCO CONFIDENTIAL**

```

 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library2>
- <Library3>
 <Name>configured.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library3>
- <Library4>
 <Name>populated.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library4>
- <Library5>
 <Name>sosd.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library5>
- <Library6>
 <Name>statusd.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library6>
- <Library7>
 <Name>pixd.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library7>
- <Library8>
 <Name>translationd.dll</Name>
 <Location>c:\ismg\PIX\libs</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <PIX />
 </References>
</Library8>
- <Custom1>
 <Name>CustomResourceOne</Name>
 <Location>d:\ismg\core</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <Core />
 </References>
</Custom1>
- <Custom1>
 <Name>CustomResourceTwo</Name>
 <Location>d:\ismg\core</Location>

```

**CISCO CONFIDENTIAL**

```

 <ReferenceCount>1</ReferenceCount>
 - <References>
 <Core />
 </References>
</Custom1>
- <Custom2>
 <Name>CustomResourceThree</Name>
 <Location>d:\ismg\core</Location>
 <ReferenceCount>1</ReferenceCount>
 - <References>
 <Core />
 </References>
</Custom2>
</Resources>
- <Core>
 <Location>d:\ismg\core</Location>
 <Configurations />
 <Libraries />
 - <ExtensionLibraries>
 <Library1 />
 </ExtensionLibraries>
 <InitializationInfo />
- <Custom>
 <Custom1 />
 <Custom1 />
 <Custom2 />
</Custom>
<Java />
<Notifications />
- <Logging>
 <Location>d:\ismg\core\log\core.log</Location>
 <axiom priority="DEBUG" />
 <eta priority="DEBUG" />
 <coreagent priority="DEBUG" />
</Logging>
</Core>
- <PIX>
 <Location>d:\ismg\PIX</Location>
 <Configurations />
 <Libraries />
 - <ExtensionLibraries>
 <Library2 />
 <Library3 />
 <Library4 />
 <Library5 />
 <Library6 />
 <Library7 />
 <Library8 />
 </ExtensionLibraries>
 <InitializationInfo />
 <Custom />
 <Java />
 <Notifications />
 <Logging />
</PIX>
</CCRRoot>

```

**CISCO CONFIDENTIAL**

## CCREntry Data Structure

`CCR::CCREntry` are strings that contain the information to specify an entry as well as string-based custom key value pairs.

```
std::string rootelement;
std::string subelement;
std::string resourcetype;
std::string resourcedata;
std::string resourcename;
std::string resourcelocation;
```

## CCRResponse Data Structure

`CCR::CCRResponse` is a simple data structure that the `CCRInterface` will return that describes the result of a function execution.

```
int responsecode;
std::string description;
std::list<CCREntry*> returnedValues;

static final int SUCCESS = 0;
static final int FAILURE = 1;
static final int EXISTS = 2;
```

## Using the CCR C++ API

This topic describes the functions for these components:

- [CCRInterface Functions](#)
- [CCREntry Functions](#)
- [CCRResponse Functions](#)

For the corresponding Java functions, see the [“Using the CCR Java Interface”](#) section on page 13-51.

## CCRInterface Functions

The functions and fields of the `CCRInterface` component are:

```
Cn::SharedPtr<CCRProcess> theProcess
```

The process to which the interface will talk.

```
CCRInterface::CCRInterface()
```

Creates an interface. The `CCRProcess::StartProcess()` function will be called in order to retrieve the process.

```
CCRInterface::CCRInterface(std::string fileName)
```

Creates an interface. The `CCRProcess::StartProcess(fileName)` function will be called in order to retrieve the process.

**CISCO CONFIDENTIAL**

**CCRInterface::CCRInterface(Cn::SharedPtr<CCRProcess> theProcess)**

Creates an interface with an already existing process. As the interface should be the only one connecting to the process, this constructor might not be necessary, desired, or needed.

**CCRResponse CCRInterface::addEntry(CCREntry\* theEntry)**

Adds a new entry to the registration repository. If the entry exists, its reference count gets incremented. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The new entry to be added.
Return	CCRResponse	The response to the addEntry request.

**CCRResponse CCRInterface::removeEntry(CCREntry\* theEntry)**

Removes an entry from the repository. If it is the last reference to that entry, it is removed. Otherwise, the reference count is decremented. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to be removed.
Return	CCRResponse	The response to the removeEntry request.

**CCRResponse CCRInterface::updateEntry(CCREntry\* theEntry, CCREntry\* newEntry)**

Updates an entry in the repository. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to be updated.
newEntry	CCREntry*	The new entry information. Existing references should be checked and reference counts should be decremented/ incremented accordingly.
Return	CCRResponse	The response to the updateEntry request.

**CCRResponse CCRInterface::retrieveEntry(CCREntry\* theEntry)**

Retrieves the complete info for an entry. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to retrieve.
Return	CCRResponse	The response to the retrieveEntry request.

**CCRResponse CCRInterface::retrieveEntriesOfType(CCREntry\* theEntry)**

Retrieves all entries of the specific type. If there is no rootelement value, it should return the entries from the resources directory. It takes the following parameters:



**CISCO CONFIDENTIAL**

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry type to return.
Return	CCRResponse	The response to the retrieveEntriesOfType request.

**CCRResponse CCRInterface::applyTomcatConfiguration(CCREntry\* theEntry)**

Will apply the `tomcatconfig` entry to the appropriate Tomcat config file. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry that describes the proper <code>tomcatconfig</code> .
Return	CCRResponse	The response to the <code>applyTomcatConfiguration</code> request.

**CCRResponse CCRInterface::removeTomcatConfiguration(CCREntry\* theEntry)**

Will remove the `tomcatconfig` entry from the appropriate Tomcat config file. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry that describes the proper <code>tomcatconfig</code> .
Return	CCRResponse	The response to the <code>removeTomcatConfiguration</code> request.

**boolean CCRInterface::entryExists(CCREntry\* theEntry)**

Determines whether or not an entry already exists in the repository. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to examine.
Return	boolean	True if it exists. Otherwise, false.

**int CCRInterface::getEntryReferenceCount(CCREntry\* theEntry)**

Determines the number of reference counts of a particular entry. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCREntry*	The entry to examine.
Return	int	The number of references of the entry.

**CCRResponse\* CCRInterface::addNotification(std::string mdc, std::string location, std::string protocol)**

Adds a notification entry to an MDC in the CCR. It takes the following parameters:

**CISCO CONFIDENTIAL**

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the new notification.
location	std::string	The notification source.
protocol	std::string	The protocol that the notification will use.
Return	CCRResponse	The response to the addEntry request.

**CCRResponse\* CCRInterface::addMDC(CCEntry\* theEntry)**

Adds a new MDC to the CCR. It takes the following parameters:

Parameter Name	Type	Purpose
theEntry	CCEntry*	The new MDC to be added.
Return	CCRResponse	The response to the addMDC request.

**CCRResponse\* CCRInterface::setLoggingLocation(std::string mdc, std::string location)**

Sets the location and file name for logging messages within an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the new logging location.
location	std::string	The logging source. Should include the file name and full path.
Return	CCRResponse	The response to the setLoggingLocation request.

**CCRResponse\* CCRInterface::addLoggingCategory(std::string mdc, std::string name, std::string priority)**

Adds a new category for logging within an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the new logging category.
name	std::string	The name of the new category.
priority	std::string	The priority that the new category will use.
Return	CCRResponse	The response to the addLoggingCategory request.

**CCRResponse\* CCRInterface::updateLoggingCategory(std::string mdc, std::string name, std::string priority)**

Updates a category's priority within an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have the updated logging category.
name	std::string	The name of the category.

**CISCO CONFIDENTIAL**

Parameter Name	Type	Purpose
priority	std::string	The priority to which the category will be updated.
Return	CCRResponse	The response to the updateLoggingCategory request.

**std::string CCRInterface::getClassPath(std::string MDCName)**

Returns a ";" separated classpath for an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
MDCName	std::string	The MDC that will have its classpath returned.
Return	std::string	A ";" separated list of Java libraries and directories.

**std::string CCRInterface::getLibraryPath(std::string MDCName)**

Returns a ";" separated path for an MDC's libraries. It takes the following parameters:

Parameter Name	Type	Purpose
MDCName	std::string	The MDC that will have its library path returned.
Return	std::string	A ";" separated list of libraries.

**std::string CCRInterface::getExtensionLibraryNames()**

Returns a ";" separated list of all of the extension libraries, used primarily for loading purposes. It takes the following parameter:

Parameter Name	Type	Purpose
Return	std::string	A ";" separated list of libraries.

**std::string CCRInterface::getMDCNames()**

Returns a ";" separated list of the MDC names within the CCR. It takes the following parameter:

Parameter Name	Type	Purpose
Return	std::string	A ";" separated list of MDC names.

**std::string CCRInterface::getMDCLoggingCategories(std::string mdc)**

Returns a ";" separated list of all logging categories of an MDC. It takes the following parameters:

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have its logging categories returned.
Return	std::string	A ";" separated list of of logging categories for an MDC.

**std::string CCRInterface::getLoggingLocation(std::string mdc)**

Returns the logging location for an MDC. It takes the following parameters:

**CISCO CONFIDENTIAL**

Parameter Name	Type	Purpose
mdc	std::string	The MDC that will have its logging location returned.
Return	std::string	A MDC's logging location.

```
void CCRInterface::writeTree()
```

Writes and saves the CCR.

## CCREntry Functions

The functions and fields of CCREntry are:

```
private std::string rootElement
```

Describes the root element that contains the entry.

```
private std::string subElement
```

Describes the sub element that contains the entry.

```
private std::string resourceType
```

Describes the type of the resource.

```
private std::string resourceData
```

Describes the data that the resource might contain.

```
private std::string resourceName
```

Describes the name of the resource.

```
private std::string resourceLocation
```

Describes the location of the resource.

```
private CustomTagTable customTags
```

A hashtable that describes any custom keys and values of the resource.

```
private PasswordTable customPwds
```

A hashtable that describes the passwords for any encrypted custom keys.

```
6. private std::string m_dataEncryptPwd
```

The password for an encrypted data value.

```
7. private std::string m_locationEncryptPwd
```

The password for an encrypted location value.

```
8. private std::string m_nameEncryptPwd
```

The password for an encrypted name value.

```
9. CCREntry::CCREntry()
```

Default constructor that sets all of the values of the entry to "".

**CISCO CONFIDENTIAL**

10. `CCREntry::CCREntry(std::string stringForm)`

Constructor that creates a `CCREntry` from one that was created to a `std::string` using the `toString` method.

11. `CCREntry::CCREntry(std::string rootelement, std::string subelement, std::string resourcetype, std::string resourcedata)`

Constructor that sets all of the values of the entry to the argument's values.

Parameter Name	Type	Purpose
rootelement	<code>std::string</code>	The new value of the root element.
subelement	<code>std::string</code>	The new value of the sub element.
resourcetype	<code>std::string</code>	The new value of the resource type.
resourcedata	<code>std::string</code>	The new value of the resource data.

12. `std::string CCREntry::getRootElement()`

Getter function for the root element value.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	The root element value.

13. `std::string CCREntry::getSubElement()`

Getter function for the sub element value.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	Returns the sub element value.

14. `std::string CCREntry::getResourceType()`

Getter function for the resource type value.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	Returns the resource type value.

15. `std::string CCREntry::getResourceData()`

Getter function for the resource data value.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	The resource data value.

**CISCO CONFIDENTIAL**

16. `std::string CCREntry::getResourceLocation()`

Getter function for the resource location value.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	The resource location value.

17. `std::string CCREntry::getResourceName()`

Getter function for the resource name value.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	The resource name value.

18. `CustomTagTable CCREntry::getTagTable()`

Getter function for the resource custom tags and values (in a hashtable).

Parameter Name	Type	Purpose
Return	<code>CustomTagTable</code>	The resource hashtable of custom tags and values.

19. `bool CCREntry::isDataEncrypted()`

Determines whether the data value has been encrypted.

Parameter Name	Type	Purpose
Return	<code>bool</code>	Indicates whether or not the data value is an encrypted value.

20. `bool CCREntry::isLocationEncrypted()`

Determines whether the location value has been encrypted.

Parameter Name	Type	Purpose
Return	<code>bool</code>	Indicates whether or not the location value is an encrypted value.

21. `bool CCREntry::isNameEncrypted()`

Determines whether the name value has been encrypted.

Parameter Name	Type	Purpose
Return	<code>bool</code>	Indicates whether or not the name value is an encrypted value.

**CISCO CONFIDENTIAL**

22. `bool CCREntry::isCustomKeyValueEncrypted (std::string key)`

Determines whether the value reference by the key has been encrypted.

Parameter Name	Type	Purpose
key	std::string	The key of the value to be checked.
Return	bool	Indicates whether or not the value reference by the key has been encrypted

23. `CCREntry::setRootElement (std::string value)`

Setter function for the root element value.

Parameter Name	Type	Purpose
value	std::string	The new root element value.

24. `CCREntry::setSubElement (std::string value)`

Setter function for the sub element value.

Parameter Name	Type	Purpose
value	std::string	The new sub element value.

25. `CCREntry::setResourceType (std::string value)`

Setter function for the resource type value.

Parameter Name	Type	Purpose
value	std::string	The new resource type value.

26. `CCREntry::setResourceData (std::string value)`

Setter function for the resource data value.

Parameter Name	Type	Purpose
value	std::string	The new resource data value.

27. `CCREntry::setResourceLocation (std::string value)`

Setter function for the resource location value.

Parameter Name	Type	Purpose
value	std::string	The new resource location value.

**CISCO CONFIDENTIAL**

28. `CCREntry::setResourceName(std::string value)`

Setter function for the resource name value.

Parameter Name	Type	Purpose
value	<code>std::string</code>	The new resource name value.

29. `CCREntry::setDataEncrypt(std::string value)`

Setter function for the resource data value encryption password.

Parameter Name	Type	Purpose
value	<code>std::string</code>	The new resource data value encryption password.

30. `CCREntry::setLocationEncrypt(std::string value)`

Setter function for the resource location value encryption password.

Parameter Name	Type	Purpose
value	<code>std::string</code>	The new resource name value encryption password.

31. `CCREntry::setNameEncrypt(std::string value)`

Setter function for the resource name value encryption password.

Parameter Name	Type	Purpose
value	<code>std::string</code>	The new resource name value encryption password.

32. `int CCREntry::getCustomTagCount()`

Returns the number of custom tags in the resource.

Parameter Name	Type	Purpose
return	<code>int</code>	The number of custom tags.

33. `std::string CCREntry::getCustomTagKey(int index)`

Returns the custom key at the index.

Parameter Name	Type	Purpose
index	<code>int</code>	The location of the key.
return	<code>std::string</code>	The key value.



**CISCO CONFIDENTIAL**

**34.** `std::string CCREntry::getCustomTagEntry(int index)`

Returns the custom value at the index.

Parameter Name	Type	Purpose
index	int	The location of the key.
return	std::string	The value.

**35.** `CCREntry::addCustomTag(std::string key, std::string value, std::string password = "")`

Setter for the resource data value. If the password value is not "", then that value is encrypted.

Parameter Name	Type	Purpose
key	std::string	The new resource custom key value.
value	std::string	The new resource custom value.

**36.** `std::string CCREntry::toString ()`

Converts an entry to a string representation. Used for the JNI interface.

Parameter Name	Type	Purpose
return	std::string	A string representation.

**37.** `int CCREntry::encryptName ()`

Encrypts the name value.

Parameter Name	Type	Purpose
return	int	The length of the string that was encrypted.

**38.** `int CCREntry::encryptData ()`

Encrypts the data value.

Parameter Name	Type	Purpose
return	int	The length of the string that was encrypted.

**39.** `int CCREntry::encryptLocation ()`

Encrypts the location value.

Parameter Name	Type	Purpose
return	int	The length of the string that was encrypted..

**CISCO CONFIDENTIAL**

40. `int CCREntry::encryptCustomKeyValue(std::string key)`

Encrypts the key's value.

Parameter Name	Type	Purpose
key	<code>std::string</code>	The key whose value will be encrypted.
return	<code>int</code>	The length of the string that was encrypted..

41. `std::string encrypt(std::string toEncrypt, std::string password)`

Converts a string to an encrypted value based on the password.

Parameter Name	Type	Purpose
toEncrypt	<code>std::string</code>	The string to be encrypted.
password	<code>std::string</code>	The password to the will be the encryption key.
return	<code>std::string</code>	A string representation.

42. `std::string CCREntry::decryptData(std::string toDecrypt, int size)`

Decrypts a string based on the data password key.

Parameter Name	Type	Purpose
toDecrypt	<code>std::string</code>	The string to decrypt.
size	<code>int</code>	The intended size of the decrypted string.
return	<code>std::string</code>	A decrypted string.

43. `std::string CCREntry::decryptName(std::string toDecrypt, int size)`

Decrypts a string based on the name password key.

Parameter Name	Type	Purpose
toDecrypt	<code>std::string</code>	The string to decrypt.
size	<code>int</code>	The intended size of the decrypted string.
return	<code>std::string</code>	A decrypted string.

44. `std::string CCREntry::decryptLocation(std::string toDecrypt, int size)`

Decrypts a string based on the location password key.

Parameter Name	Type	Purpose
toDecrypt	<code>std::string</code>	The string to decrypt.
size	<code>int</code>	The intended size of the decrypted string.
return	<code>std::string</code>	A decrypted string.

**CISCO CONFIDENTIAL**

45. `std::string CCREntry::decryptKeyValue(std::string toDecrypt, std::string key, int size)`

Decrypts a string based on the key's password key.

Parameter Name	Type	Purpose
toDecrypt	std::string	The string to decrypt.
key	std::string	The key whose password key will be used.
size	int	The intended size of the decrypted string.
return	std::string	A decrypted string.

## CCRResponse Functions

The functions and fields of CCRResponse are:

**private std::string description**

Provide the response's description.

**private id responseID**

Describes the ID of the response. Possible values: SUCCESS, FAILURE, or EXISTS.

**std::vector<CCREntry\*> returnedValues;**

Describes the entry or entries that returned with the response.

**static final int SUCCESS = 0;**

The ID of a successful response.

**static final int FAILURE = 1;**

The ID of an unsuccessful response.

**static final int EXISTS = 2;**

The ID of a response where the entry exists.

**CCRResponse::CCRResponse()**

Default constructor. The response ID and description are not set; use the setter functions.

**CCRResponse::CCRResponse(int type, std::string description)**

Constructor that sets the ID and description.

Parameter Name	Type	Purpose
type	int	The response's type. Possible values: SUCCESS, FAILURE, or EXISTS.
description	std::string	The response's description. Explains the type.

**CISCO CONFIDENTIAL**

**CCRResponse::CCRResponse(std::string stringForm)**

Constructor that creates a response object from one that was converted into a string.

Parameter Name	Type	Purpose
stringForm	std::string	The representation of a CCRResponse object in a string form.

**std::string CCRResponse::getDescription()**

The getter function for the description value.

Parameter Name	Type	Purpose
Return	std::string	Returns the description value.

**46. int CCRResponse::getResponseID()**

The getter function for the response's type value.

Parameter Name	Type	Purpose
Return	int	Returns the type value.

**47. std::vector<CCREntry\*> CCRResponse::getReturnedValues()**

The getter function for the response's returned entries.

Parameter Name	Type	Purpose
Return	std::vector<CCREntry*>	Returns the entries that are part of the response.

**48. CCRResponse::setDescription(std::string description)**

The setter function for the description value.

Parameter Name	Type	Purpose
description	std::string	The new value of the description.

**49. CCRResponse::setResponseID(int type)**

The setter function for the type value.

Parameter Name	Type	Purpose
type	int	The new value of the type.

**50. CCRResponse::setReturnedValues(std::vector<CCREntry\*> values)**

The setter function for the type value.

**CISCO CONFIDENTIAL**

Parameter Name	Type	Purpose
values	<code>std::vector&lt;CCREntry*&gt;</code>	The new value of the entries that belong with the response.

51. `CCRResponse::addReturnedValue(CCREntry* value)`

The setter function for the type value.

Parameter Name	Type	Purpose
value	<code>CCREntry*</code>	A new <code>CCREntry</code> to be added with the response.

52. `boolean CCRResponse::success()`

A convenience function that will tell the caller whether the response is a success response or not.

Parameter Name	Type	Purpose
Return	<code>boolean</code>	True if the response type is <code>SUCCESS</code> . Otherwise, false.

53. `boolean CCRResponse::failure()`

A convenience function that will tell the caller whether the response is a failure response or not.

Parameter Name	Type	Purpose
Return	<code>boolean</code>	True if the response type is <code>FAILURE</code> . Otherwise, false.

54. `boolean CCRResponse::exists()`

A convenience function that will tell the caller whether the response is a exists response or not.

Parameter Name	Type	Purpose
Return	<code>boolean</code>	True if the response type is <code>EXISTS</code> . Otherwise, false.

55. `std::string CCRResponse::toString()`

Converts a `CCRResponse` object to a string form. Can be reconstituted using the constructor that takes a string as an argument.

Parameter Name	Type	Purpose
Return	<code>std::string</code>	The string form of the <code>CCRResponse</code> object.

**CISCO CONFIDENTIAL**

# Using the CCR API: Example

The following procedure shows how you can use the CCR API to modify and access the CCR.

**Step 1** Initialize the CCR.

To access and modify the CCR, you must create a CCRInterface object. When this object is constructed, either an existing CCR will be loaded, or a new one will be created.

```
CCRInterface* theCCRInterface = new CCRInterface();
```

**Step 2** Create an MDC.

In order to create a new MDC, first a CCREntry object should be created where the root element is the name of the MDC, and the location value is where the MDC is located. Then the addMDC method is called on the CCRInterface object.

```
CCREntry* newMDCEntry = new CCREntry();
newMDCEntry->setRootElement("NewMDCName");
newMDCEntry->setResourceLocation("d:\mdclocation");
theCCRInterface->addMDC(newMDCEntry);
```

**Step 3** Create entries for extension libraries.

When the MDC is created, extension libraries pertaining to the MDC must be added. They are located in the MDC element and under the ExtensionLibraries element. The name of each entry will be the name of the library, and the location of the entry is where the library is located. Also, the entry type will be set to Library.

```
CCREntry* newExtensionLibrary = new CCREntry();
newExtensionLibrary->setRootElement("NewMDCName");
newExtensionLibrary->setSubElement("ExtensionLibraries");
newExtensionLibrary->setResourceName("mylibrary.dll");
newExtensionLibrary->setResourceType("Library");
theCCRInterface->addEntry(newExtensionLibrary);
```

**Step 4** Add a new logging location for the MDC.

You must specify the MDC that will be changed as well as the complete path of the file in which you want to track the logs.

```
theCCRInterface->setLoggingLocation("TheMDC", "d:\mymdc\logs\mymdc.log");
```

**Step 5** Add a new logging entry for the MDC.

You must add the MDC name, the category name, and the priority level. Priority levels can be DEBUG, INFO, WARN, ERROR, or FATAL.

```
theCCRInterface->addLoggingCategory("TheMDC", "NewLoggingCategory", "DEBUG");
```

**CISCO CONFIDENTIAL**

# Using the CCRAccess Client

CCRAccess is a command-line client that allows easy initialization and manipulation of the CCR. This is an executable and should be the primary way to enter information into CCR. To use CCRAccess, enter the following on the command line:

```
CCRAccess [-options] [-commands]
```

Where:

[-options] is one or more of the options or wildcards shown in [Table 13-1](#).

[-commands] is one of the commands shown in [Table 13-2](#).

Note that this client is also available as a DLL (see “Using the CCRAccess DLL” section on page 13-50).

**Table 13-1 CCRAccess Options and Wildcards**

Option/Wildcard	Description
? -help	Print this help message.
-new	Create a new Registration Daemon (CCR).
-location <file>	Specify an existing CCR saved in a file.
-script <file>	Load commands for the CCR from a script file.
/e:<encryptionkey>	Encrypts data, name, location, or custom tag value of resource. If the key is in a file, <code>encryptionkey</code> should be <code>file:&lt;filename&gt;</code> . Otherwise, enter the string. This should be directly before the data, location, or name that you want to encrypt or before the tag of the custom value that you want to encrypt.
/d:<encryptionkey>	Decrypts data, name, location, or custom tag value of resource. If the key is in a file, <code>encryptionkey</code> should be <code>file:&lt;filename&gt;</code> . Otherwise, enter the string. This should be directly before the data, location, or name that you want to decrypt or before the tag of the custom value that you want to decrypt.

**Table 13-2 CCRAccess Commands**

Command	Description
-addResource	Adds a resource. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-addNotification	Adds a new notification entry to the CCR. This command must be followed by <MDCName> <Location> <Protocol>. No empty strings are allowed.
-addLog	Adds a new logging entry to the CCR. This command must be followed by <MDCName> <Name> <Priority>. No empty strings are allowed.
-addLogLocation	Adds the log location for an MDC. This command must be followed by <MDCName> <Location>. No empty strings are allowed.
-updateLog	Updates a new logging entry to the CCR. This command must be followed by <MDCName> <Name> <Priority>. No empty strings are allowed.
-addMDC	Adds a new MDC to the CCR. This command must be followed by <MDCName> <MDCLocation>. No empty strings are allowed.

**CISCO CONFIDENTIAL****Table 13-2 CCRAccess Commands (continued)**

<b>Command</b>	<b>Description</b>
-removeMDC	Removes an MDC from the CCR. This command must be followed by <MDCName>. No empty strings are allowed.
-removeResource	Removes a resource. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getClasspath	Gets the classpath of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getLibraryPath	Gets the library path of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getLogCategories	Gets the logging categories of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getLogLocation	Gets the logging file location of the MDC. This command must be followed by <MDCName>. No empty strings are allowed.
-getELNames	Gets the extension libraries in CCR.
-getMDCNames	Gets the name of the MDC in CCR.
-getResource	Retrieves a resource. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getData	Retrieves a resource's data. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getLocation	Retrieves a resource's location. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getName	Retrieves a resource's name. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getCustom	Retrieves a resource's custom tags and values. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getCustomTagValue	Retrieves a resource's tag's value. This command must be followed by <TagName> <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".



**CISCO CONFIDENTIAL****Table 13-2 CCRAccess Commands (continued)**

Command	Description
-getResourcesOfType	Retrieves a group of resources. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-entryExists	Checks if a resource exists. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".
-getEntryRefCount	Gets the number of MDCs that reference the resource. This command must be followed by <RootElement> <SubElement> <ResourceType> <ResourceData> <ResourceLocation> <ResourceName>. Custom values and keys can be added at the end. They must be added in pairs. Empty strings should be entered as "".

## Scripting CCRAccess

CCRAccess can load and run scripts, allowing you to perform “bulk” manipulation of the CCR. Scripts are simple, separated lines, with each line a CCRAccess command or option.

For example:

```
-addMDC NewMDCName d:\mymdcllocation;
-addResource NewMDCName ExtensionLibraries Library EMPTYSTRING d:\mymdcllocation\libs
mylibrary.dll;
-addLogLocation TheMDC d:\mymdc\logs\mymdc.log;
-addLog TheMDC NewLoggingCategory DEBUG
```

This information can be stored in a file called `mdcsetup.script`. The following procedure is an example illustrating how you can use the CCRAccess command line interface and its scripting capabilities.

---

**Step 1** Use CCRAccess commands. For example:

```
CCRAccess -addMDC NewMDCName d:\mymdcllocation;
CCRAccess -addResource NewMDCName ExtensionLibraries Library EMPTYSTRING
d:\mymdcllocation\libs mylibrary.dll;
CCRAccess -addLogLocation TheMDC d:\mymdc\logs\mymdc.log;
CCRAccess -addLog TheMDC NewLoggingCategory DEBUG
```

**Step 2** Load a script. For example:

```
CCRAccess -script mdcsetup.script
```

---

**CISCO CONFIDENTIAL**

# Using the CCRAccess DLL

CCR includes `ccraccess.dll`, a dynamic link library that provides all of the same functionality as the `ccraccess.exe` client described in the “Using the CCRAccess Client” section on page 13-47. The DLL version is significantly quicker to call (by approximately four seconds per call) than the EXE version. CCRAccess is often called during product installations, so developers who need to trim install times will find it useful to replace existing `ccraccess.exe` calls with `ccraccess.dll` calls.

The CCRAccess DLL interface is defined with the following prototype:

```
int EXPORTED ccraccess(char* commandLineInputString, char* resultFileName);
```

Where:

- `commandLineInputString` is one or more of the command-line arguments available for `ccraccess.exe`, which are listed in Table 13-2 on page 13-47. Multiple commands can be distinguished using the `|` variable separator
- `resultFileName` is the name of the file where the result of execution is to be dumped.

Calls to `ccraccess.dll` always return a zero on success and non-zero on failure.

Follow the steps below to call `ccraccess.dll` from a Windows-platform install (rul) file. Example 13-1 shows sample code that follows all of these steps.

- 
- Step 1** Prepare a `commandLineInputString` parameter by setting it to a string containing the previously used `ccraccess.exe`'s command line parameters. Concatenate each parameter with `|` in place of the space separator. For example: `"-addResource|Core|Custom|Custom|... "`
- Step 2** Prepare a `resultFileName` parameter by setting it to an empty string (if you are writing into CCR using `-addResource`) or to a valid file name (if you will be reading from CCR using `-getResource`).
- Step 3** Load the library by calling the install library's utility function `loadCcrDll()`. Check for a zero return value to confirm that CCRlibrary is loaded properly.
- Step 4** Make the call: `return_value = ccraccess.ccraccess(commandLineInputString, resultFileName);`, where `return_value` holds the status of the call.
- Step 5** If you are reading from CCR using `-getResource`, parse the output file to retrieve the results.
- Step 6** Unload the library using the install library's utility function `UnloadCcrDll()`.
- 

**Example 13-1 Using ccraccess.dll**

```
function MODULENAME_postinstall()
{

#define CMD_CCR_CUSTOM_PREFIX_DLL "-addResource|Core|Custom|Custom"
#define CMD_CCR_REG_HTTP_PORT_DLL
CMD_CCR_CUSTOM_PREFIX_DLL+"|"+G_Port+"|"+EMPTYSTRING|HttpPort"
STRING resultFileName;

//loading CCRDLL and reporting in case of error

if (loadCcrDll() != 0) then
 szTit="CCRaccess Dll's Not Found";
 SetDialogTitle (DLG_MSG_INFORMATION,szTit);
```

**CISCO CONFIDENTIAL**

```

 MessageBox("ccraccess Dll's are not found. Installation will
abort", SEVERE);
 SureDeleteFile(WINDISK+"\\CMFLOCK.TXT");
 abort;
 endif;

// Calling CCRDLL API to set
nCcrVal=ccraccess.ccraccess(CMD_CCR_REG_HTTP_PORT_DLL, resultFileName);

// More lines using the ccraccess.dll.

UnloadCcrDll();
}

```

## Using the CCR Java Interface

The Java interface for CCR is virtually identical to the functions and fields described in the “[Using the CCR C++ API](#)” section on page 13-31. However, it does rely heavily on the `toString()` functionality of `CCRResponse` and `CCREntry`. Java users should use `com.cisco.core.ccr.CCRInterface` when manipulating CCR.

**com.cisco.core.ccr.CCrentry**

This class is identical to the `CCREntry` C++ class, with obvious language differences. The changes are as follows:

- `std::string` changes to `java.lang.String`
- `CustomTagTable` and `PasswordTable` change to `java.util.Hashtable`
- `std:vector` changes to `java.util.Vector`

Encryption is slightly different as the algorithms for this are on the C++ side. `CCRInterface` provides methods to handle this.

**com.cisco.core.ccr.CCRInterface**

This class is identical to the `CCRInterface` C++ class with obvious language differences.

The changes are as follows:

- `std::string` changes to `java.lang.String`
- `CustomTagTable` and `PasswordTable` change to `java.util.Hashtable`
- `std:vector` changes to `java.util.Vector`

Access to the CCR via Java should be done through this class.

There are some additional methods to handle decryption of a `CCREntry`:

- `public CCREntry decryptData(CCREntry entry, int size)`

This method will decrypt the data value based on the entry’s data password. You will need to know the size of the original data value.

- `public CCREntry decryptName(CCREntry entry, int size)`

This method will decrypt the name value based on the entry’s name password. You will need to know the size of the original name value.

**CISCO CONFIDENTIAL**

- `public CCREntry decryptLocation(CCREntry entry, int size)`

This method will decrypt the location value based on the entry's location password. You will need to know the size of the original location value.

- `public CCREntry decryptKeyValue(CCREntry entry, String key, int size)`

This method will decrypt the key's value based on the entry's key password. You will need to know the size of the original key value.

**com.cisco.core.ccr.CCRResponse**

This class is identical to the CCRResponse C++ class, with obvious language differences. The changes are as follows:

- `std::string` changes to `java.lang.String`
- `CustomTagTable` and `PasswordTable` change to `java.util.Hashtable`
- `std:vector` changes to `java.util.Vector`

**JNICCRInterface**

`com.cisco.core.ccr.CCRInterface` uses this class to bridge the Java/C++ gap via JNI. It is possible to use this class for the interface, but it is recommended that you use the `CCRInterface` class in `com.cisco.core.ccr`.

## Encrypting and Decrypting CCREntry Values

You can encrypt and decrypt CCREntry name, data, location, and custom entry values. You must provide a key in the form of a file name or string value in order to encrypt or decrypt a value. The following topics show how to perform encryption and decryption using all of the available access methods:

- [Encrypting Entry Names](#)
- [Decrypting Entry Names](#)
- [Encrypting Entry Data](#)
- [Decrypting Entry Data](#)
- [Encrypting Entry Locations](#)
- [Decrypting Entry Locations](#)
- [Encrypting Custom Entries](#)
- [Decrypting Custom Entries](#)

## Encrypting Entry Names

You can encrypt entry names via the C++, Java, or CCRAccess interfaces.

To encrypt an entry name using the C++ interface:

- 
- Step 1** Create a valid CCREntry.
  - Step 2** Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value.
  - Step 3** Call `addEntry` via `CCRInterface`.

## CISCO CONFIDENTIAL

The entry will be added to the CCR with its name value encrypted.

---

To encrypt an entry name using the Java interface:

---

- Step 1** Create a valid CCREntry.
- Step 2** Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value.
- Step 3** Call `addEntry` via `CCRInterface`.

The entry will be added to the CCR with its name value encrypted.

---

To encrypt an entry name using `CCRAccess`:

---

- Step 1** Consider your key to be “mykey”.
- Step 2** Call:

```
CCRAccess -addResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation /e:mykey MyResourceName <Any custom tag pairs>
```

The entry will be added to the CCR with its name value encrypted.

---

## Decrypting Entry Names

You can decrypt entry names via the C++, Java, or `CCRAccess` interfaces.

To decrypt an entry name using the C++ interface:

---

- Step 1** Create a valid CCREntry.
- Step 2** Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value. The name value should be “”.
- Step 3** Call `retrieveEntry` via `CCRInterface`.

The entry retrieved will have the decrypted name value if the password was correct.

---

To decrypt an entry name using the Java interface:

---

- Step 1** Create a valid CCREntry.
- Step 2** Call `setNameEncrypt` with the key (password) with which you want to encrypt the name value. The name value should be “”.
- Step 3** Call `retrieveEntry` via `CCRInterface`.

The entry retrieved will have the decrypted name value if the password was correct.

---

**CISCO CONFIDENTIAL**

To decrypt an entry name using CCRAccess:

---

**Step 1** Consider your key to be “mykey”.

**Step 2** Call:

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation /d:mykey "" <Any custom tag pairs>
```

The entry retrieved will have the decrypted name value if the password was correct.

---

## Encrypting Entry Data

You can encrypt entry data via the C++, Java, or CCRAccess interfaces.

To encrypt an entry’s data using the C++ interface:

---

**Step 1** Create a valid CCREntry.

**Step 2** Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value.

**Step 3** Call `addEntry` via the CCRInterface.

The entry will be added to the CCR with its data value encrypted.

---

To encrypt an entry’s data using the Java interface:

---

**Step 1** Create a valid CCREntry.

**Step 2** Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value.

**Step 3** Call `addEntry` via CCRInterface.

The entry will be added to the CCR with its data value encrypted.

---

To encrypt an entry’s data using CCRAccess:

---

**Step 1** Consider your key to be “mykey”.

**Step 2** Call:

```
CCRAccess -addResource MyMDC MyResourceSubdirectory MyResourceType /e:mykey MyResourceData
MyResourceLocation MyResourceName <Any custom tag pairs>
```

The entry will be added to the CCR with its data value encrypted.

---

**CISCO CONFIDENTIAL**

## Decrypting Entry Data

You can decrypt an entry's data via the C++, Java, or CCRAccess interfaces.

To decrypt an entry's data using the C++ interface:

- 
- Step 1** Create a valid CCREntry.
  - Step 2** Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value. The data value should be "".
  - Step 3** Call `retrieveEntry` via `CCRInterface`. The entry retrieved will have the decrypted data value if the password was correct.
- 

To decrypt an entry's data using the Java interface:

- 
- Step 1** Create a valid CCREntry.
  - Step 2** Call `setDataEncrypt` with the key (password) with which you want to encrypt the data value. The data value should be "".
  - Step 3** Call `retrieveEntry` via `CCRInterface`. The entry retrieved will have the decrypted data value if the password was correct.
- 

To decrypt an entry's data using CCRAccess:

- 
- Step 1** Consider your key to be "mykey".
  - Step 2** Call:  

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType /d:mykey ""
MyResourceLocation MyResourceName <Any custom tag pairs>
```
- 

The entry retrieved will have the decrypted data value if the password was correct.

---

## Encrypting Entry Locations

You can encrypt an entry's location data via the C++, Java, or CCRAccess interfaces.

To encrypt an entry's location data using the C++ interface:

- 
- Step 1** Create a valid CCREntry.
  - Step 2** Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.
  - Step 3** Call `addEntry` via `CCRInterface`. The entry will be added to the CCR with its location value encrypted.
-

**CISCO CONFIDENTIAL**

To encrypt an entry's location data using the Java interface:

- 
- Step 1** Create a valid CCREntry.
- Step 2** Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.
- Step 3** Call `addEntry` via `CCRInterface`.
- The entry will be added to the CCR with its location value encrypted.
- 

To encrypt an entry's location data using `CCRAccess`:

- 
- Step 1** Consider your key to be "mykey".
- Step 2** Call:
- ```
CCRAccess -addResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData /e:mykey
MyResourceLocation MyResourceName <Any custom tag pairs>
```
-

The entry will be added to the CCR with its location value encrypted.

Decrypting Entry Locations

You can decrypt an entry's location data via the C++, Java, or `CCRAccess` interfaces.

To decrypt an entry's location data using the C++ interface:

-
- Step 1** Create a valid CCREntry.
- Step 2** Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.
The location value should be "".
- Step 3** Call `retrieveEntry` via `CCRInterface`.
- The entry retrieved will have the decrypted location value if the password was correct.
-

To decrypt an entry's location data using the Java interface:

-
- Step 1** Create a valid CCREntry.
- Step 2** Call `setLocationEncrypt` with the key (password) with which you want to encrypt the location value.
The location value should be "".
- Step 3** Call `retrieveEntry` via `CCRInterface`.
- The entry retrieved will have the decrypted location value if the password was correct.
-

CISCO CONFIDENTIAL

To decrypt an entry's location data using CCRAccess:

Step 1 Consider your key to be “mykey”.

Step 2 Call:

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData /d:mykey
" MyResourceName <Any custom tag pairs>
```

The entry retrieved will have the decrypted location value if the password was correct.

Encrypting Custom Entries

You can encrypt custom entry data via the C++, Java, or CCRAccess interfaces.

To encrypt custom entry data using the C++ interface:

Step 1 Create a valid CCREntry.

Step 2 Call `addCustomTag` with the key (password) with which you want to encrypt the key value as the third argument.

Step 3 Call `addEntry` via `CCRInterface`.

The entry will be added to the CCR with the custom tag value encrypted.

To encrypt custom entry data using the Java interface:

Step 1 Create a valid CCREntry.

Step 2 Call `addCustomTag` with the key (password) with which you want to encrypt the key value as the third argument.

Step 3 Call `addEntry` via `CCRInterface`.

The entry will be added to the CCR with the custom tag value encrypted.

To encrypt custom entry data using CCRAccess:

Step 1 Consider your key to be “mykey”, the custom tag to be “MyCustomKey”, and the value to be “MyCustomKeyValue”.

Step 2 Call:

```
CCRAccess -addResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation MyResourceName /e:mykey MyCustomKey MyCustomKeyValue
```

The entry will be added to the CCR with its custom tag value encrypted.

CISCO CONFIDENTIAL**Decrypting Custom Entries**

You can decrypt custom entry data via the C++, Java, or CCRAccess interfaces.

To decrypt custom entry data using the C++ interface:

-
- Step 1** Create a valid CCREntry.
- Step 2** Call `addCustomTag` with the key (password) with which you want to decrypt the key value as the third argument.
The custom tag's value should be "".
- Step 3** Call `retrieveEntry` via `CCRInterface`.
The entry will be retrieved from the CCR with the custom tag value decrypted.
-

To decrypt custom entry data using the Java interface:

-
- Step 1** Create a valid CCREntry.
- Step 2** Call `addCustomTag` with the key (password) with which you want to decrypt the key value as the third argument.
The custom tag's value should be "".
- Step 3** Call `retrieveEntry` via `CCRInterface`.
The entry will be retrieved from the CCR with the custom tag value decrypted.
-

To decrypt custom entry data using CCRAccess:

-
- Step 1** Consider your key to be "mykey" and the custom tag to be `MyCustomKey MyCustomKeyValue`.
- Step 2** Call:

```
CCRAccess -getResource MyMDC MyResourceSubdirectory MyResourceType MyResourceData
MyResourceLocation MyResourceName /d:mykey MyCustomKey
""
```

The entry retrieved will have the decrypted key value if the password was correct.



CISCO CONFIDENTIAL

CHAPTER 14

Using the Device Credentials Repository

The Device Credentials Repository (DCR) is a common repository of devices, their attributes and their credentials. An DCR system consists of one or more DCR Servers that store and distribute device information, applications using DCR APIs to access this information, and the Device & Credentials Admin GUI accessible from the CiscoWorks Home Page.

When implemented, DCR provides:

- Easier, centralized access to device and credentials data.
- Secure device-data persistence, access and transport.
- Rationalized and controlled replication of device information, with less user-level reconciliation.
- Better integration with third-party and Cisco network-management applications.



Note

DCR stores device information. It does not communicate with devices directly. The code that interacts directly with devices and fetches their data for storage in DCR is the responsibility of your application.

The following topics describe how to integrate DCR with your application:

- [Understanding DCR](#)
- [Using DCR](#)

For more information about DCR, see:

- *Device List and Credentials Repository Server Functional Specification, EDCS-283571*
- *DCR Deployment Scenarios, Use Cases and Guidelines for Applications, EDCS-283285*
- *Device List and Credentials Repository Master/Slave Functional Specification, EDCS-283284*
- *Device list and Credentials Repository Import Export Software Functional Specification, EDCS-285702*



Note

CiscoWorks users know DCR by the acronym DCA, after the Device & Credentials Admin GUI they use to update DCR device lists. Some developers also use the DCA acronym to refer to DCR.

CISCO CONFIDENTIAL

Understanding DCR

All network management applications need to store basic attributes of the devices they manage. This includes device credentials, such as SNMP community strings consisting of a user ID-and-password pair.

Devices and their credentials represent a special data management problem. They must be both:

- **Shared:** Multiple copies of device data are wasteful. If there is no single device data store that can be shared, users of multiple network management applications must waste resources reconciling the data, either via manual means or via automated tools they must build and maintain.
- **Secure:** Independent copies of credentials are insecure by definition, as they must be manually copied among applications, or transmitted using insecure means. Insecure credentials are dangerous, as anyone with access to credentials may reconfigure devices in destructive ways.

The DCR solution enables multiple applications to share device lists and credentials using a client-server mechanism, with secure storage and communications.

The following topics present basic information on DCR and how it works:

- [About DCR Features and Benefits](#)
- [How DCR Works](#)
- [About the DCR Modes](#)
- [About the DCR Components](#)
- [How DCR Masters and Slaves Interact](#)
- [How DCR Secures Device and Credentials Data](#)
- [About DCR Data Storage](#)
- [Integrating DCR with OGS](#)
- [Integrating DCR with ACS](#)
- [About DCR Events](#)

About DCR Features and Benefits

DCR offers the following capabilities:

- DCR stores device attributes and credentials, permits users to attach custom data to devices, and permits default grouping of devices.
- DCR supports proxy device attributes, storage of IPv6 and SNMP v3 device and credential information, and assigns a unique, internally generated DCR Device ID to every device.
- DCR supports “unreachable devices” — devices known to be “on the shelf” and not yet deployed, or “phone home” devices in transit to their location during initial deployment. DCR allows users to import these devices and retain credentials for them even though some critical information about them may not be available at first.
- DCR supports device pre-provisioning. Applications often do not know essential attributes of pre-provisioned devices, such as their host names or IP addresses. Using the DCR `device_identity` attribute, applications can uniquely identify pre-provisioned devices.
- Clients can add, modify, and delete DCR devices using the DCA GUI. DCR also permits users to populate the database via import from many sources, and to export device data for use with third-party products like HP OpenView and Netview.

CISCO CONFIDENTIAL

- DCR uses several attributes to detect and stop the creation of duplicate devices. If a new device has (or an update causes an existing device to change to) a display_name or fully-qualified DNS name (i.e., host_name + domain_name) that is the same as an existing device, the operation will fail.
- DCR data distribution is flexible, scalable, reliable, using a network of Master/Slave servers. All server-to-server communications take place using CSTM (see [Chapter 31, “Using the Common Services Transport Mechanism”](#)).
- DCR data is secure. DCR encrypts credential data using a static encryption scheme that must be hard-coded into the methods used for encryption (considered acceptable by the security team).
- DCR communications are secure. Access to device data via API calls or the DCA GUI is performed only by secured channel (HTTPS) and authorized clients. DCR sends events to applications using ESS (see [Chapter 19, “Using Event Services Software”](#)).
- DCR supports a full range of administrative features, including the DCA GUI, change logs and reports, and full compatibility with CWCS backup and restore.

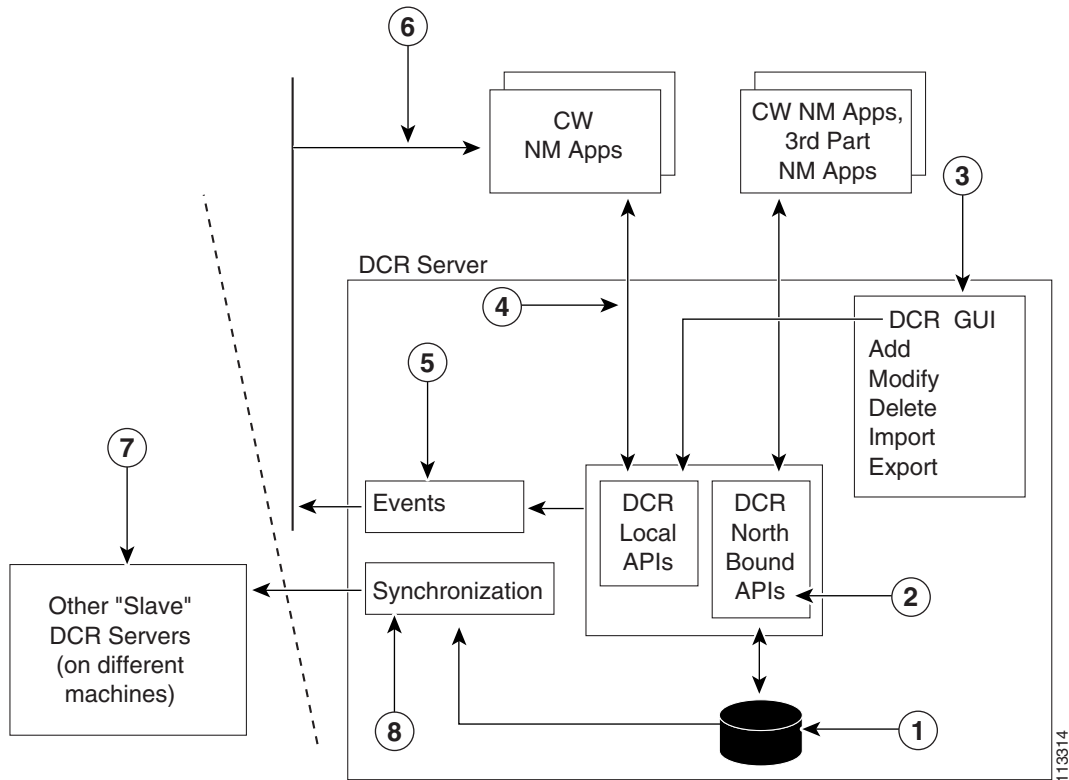
How DCR Works

Figure 14-1 shows the DCR system flow:

1. DCR provides a database structure to store device lists, the attributes for each device, and device credentials.
2. DCR provides north-bound APIs to allow remote applications to add, update and manage the data. It also provides Java APIs for local CiscoWorks network management applications.
3. A DCR Server in Master or Standalone mode (see the [“About the DCR Modes”](#) section on [page 14-4](#)) provides the DCA GUI to administer device data. Users can use this GUI to add or import devices and credentials, modify device credential data, or delete devices. Note that only the Master and Standalone modes provide access to DCA. CiscoWorks applications using a DCR Server in Slave mode cannot use DCA.
4. Cisco Works applications residing on the same machine as DCR can access or modify the data using local Java APIs (most CiscoWorks applications) or north-bound APIs (e.g., C++ CiscoWorks applications). Third-party network management applications can also access or modify the data using the secure north-bound APIs.
5. Changes in DCR data (whether the DCR Server is in Master or Slave mode) are broadcast to network management applications on that server as Event Services Software (ESS) events. Third-party applications who register with DCR can also receive HTTP- based broadcasts of these events.
6. DCR clients (any Cisco Works application) may choose to listen to these events. Upon receiving these events, the application can ignore any events that are of no interest to it (e.g, applications may want to filter out updates about devices they do not support). Upon filtering the data, the application can get the device details from DCR through APIs.
7. DCRs in Slave mode reside on other CWCS Servers installed on different machines.
8. A DCR in slave mode synchronizes its data with that of its master automatically every 45 seconds (the default) or whenever there is a change in the master data (the slave must be set to listen for and receive sync events from the master). During synchronization, the slave fetches all newly updated data from the Master and updates its local database accordingly.

CISCO CONFIDENTIAL

Figure 14-1 DCR System Flow



About the DCR Modes

Any instance of DCR can run in Master, Slave, or Standalone mode:

- **In Master mode**, DCR hosts the authoritative, or master, list of all devices and their credentials. All other DCRs in the same management domain which are running in Slave mode will normally share this list. There can be only one DCR running in Master mode in a single management domain. All Slave DCRs update the Master DCR before updating themselves, and changes to the Master DCR device data are propagated to Slave DCRs using CWCS CSTM. The Master always contains the most up-to-date device data in the management domain.
- **In Slave mode**, DCR maintains an exact replica of the data managed by the Master DCR for the management domain. There can be multiple Slave DCRs per management domain. When a client application updates device data in a Slave DCR, that Slave DCR will first send the update to the Master DCR, and then update itself from the Master. Since it is possible for a Slave to miss some updates, Slaves can also initiate synchronization with the Master DCR.
- **In Standalone mode**, DCR maintains an independent repository of device list and credential data. It does not participate in a management domain and its data is not shared with any other DCR. It does not communicate with or contain registration information about any other Master, Slave, or Standalone DCR.

CISCO CONFIDENTIAL

When considering DCR modes, remember that:

- DCR is a shared component, so there is only one DCR Server per CWCS Server. All client applications installed on that CWCS Server use the local DCR Server running on that machine, regardless of mode.
- The DCR Server mode is transparent to applications.
- DCR is not fault tolerant. It does not ensure that client applications receive update notifications, and does not validate the repository data. It relies on DCR clients to update repository data.
- DCR currently supports only one Master per management domain, with multiple Slaves. It does not support multiple Master DCR Servers in the same domain.
- All newly-installed DCR Servers start in Standalone mode by default. Users set it to Master or Slave mode using the DCA GUI or via the DCR CLI (see [“Using the DCR Command-Line Interface” section on page 14-46](#)).

About the DCR Components

An instance of DCR consists of the following components:

- **Repository:** Stores and maintains device and credentials data in the CWCS database. It includes all relevant database server functions, including the APIs providing access to the data, duplicate-device detection, and maintenance of time stamps.

The Repository also includes DCR-specific functions. These include generating the unique DCR Device IDs used to identify each device on both Masters and Slaves, information about the DCR Server’s current mode, and synchronization information.

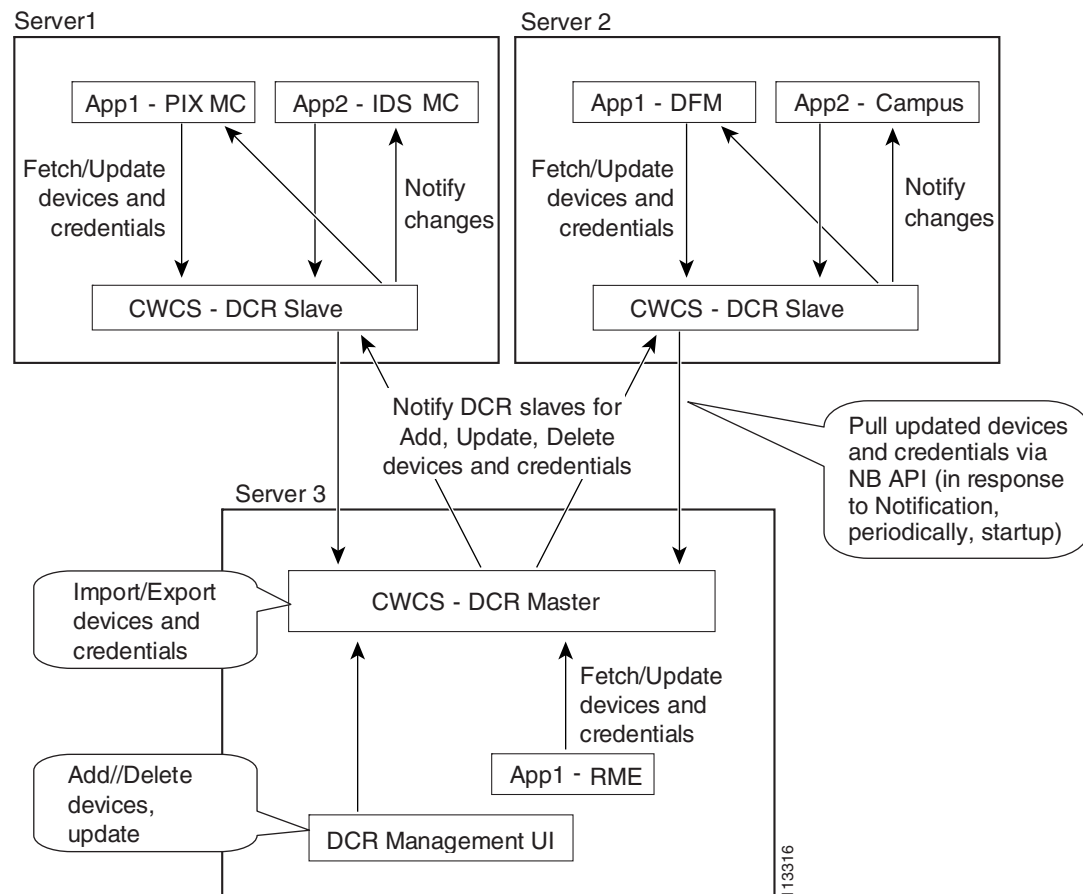
- **Registry:** Maintains the information needed to contact corresponding DCR Servers. For a Master DCR Server, it stores the URL and port number of each of its Slaves; for a DCR Slave, it stores the URL and port of the Master.
- **Notifier:** Active only on Master DCR Servers, it notifies Slaves of changes in Repository data whenever such changes occur. The Notifier uses the Registry to get the contact details for all of the Slaves. DCR uses CSTM (see [Chapter 31, “Using the Common Services Transport Mechanism”](#)) to send the notifications.
- **Receiver:** Active only on Slaves, it processes notifications from the Master about Repository data changes. To keep Slave data tightly coupled with the Master, Slaves generate events for these update notifications and send the events to applications using the Slaves (note that notifications are only exchanged among servers; applications only receive events). Applications are responsible for filtering out events for devices in which they are not interested, using the DCR Device ID (and other details, such as the sysObjectID) contained in the event.
- **Synchronizer:** Lets Slave DCRs get updates from their respective Masters on a regular basis, and in response to update notifications from the Master.
- **Event Generator:** Generates events for the applications installed on and sharing a local DCR on a single CWCS server. The Event Generator runs regardless of the DCR mode (see the [“How DCR Masters and Slaves Interact” section on page 14-6](#)). The generated events notify subscribing applications that there is modified, new or deleted device or credentials information in the repository. Each event carries enough information for the subscribing application to determine if it does or does not want the new information, and if so, to formulate a query to retrieve it. Applications are responsible for subscribing to these events, and can receive them either via ESS or HTTP (see the [“About DCR Events” section on page 14-15](#)).

CISCO CONFIDENTIAL**How DCR Masters and Slaves Interact**

Figure 14-2 shows an example of a DCR Master/Slave setup. In this example, several CiscoWorks applications are installed on different CWCS Servers. Each server has one local DCR Server. The applications installed on each CWCS Server interact with the local DCR Server only. For example: PIX MC and IDS MC on Server 1 interact with the DCR Server in Slave mode on Server 1; DFM and Campus Manager interact with the DCR Server in Slave mode on Server 2; RME interacts with the DCR Server in Master mode on Server 3.

The Master sends notifications of any changes in its repository to both Slaves. The Slaves in turn make synchronize with the Master to get its device attribute and credential updates. Both the Master and the Slaves also broadcast events to applications via ESS (they also publish events via HTTP to subscribed third-party applications).

Figure 14-2 Example DCR Master/Slave Setup



The interactions among this Master and its Slaves vary depending on the kinds of operations being performed and the DCR Server mode. The following topics explain what happens in each case:

- [How DCR Adds Devices](#)
- [How DCR Modifies Devices](#)
- [How DCR Deletes Devices](#)

CISCO CONFIDENTIAL

How DCR Adds Devices

Users whose application resides on the same CWCS server with a DCR Master can use the local DCR APIs or the DCA GUI to add a new device directly to that Master. When this happens:

1. The DCR Master checks whether the device is a duplicate or not. If it is a duplicate, the operation fails.
2. If the new device is not a duplicate, the DCR Master adds the device to the repository.
3. The DCR Master then:
 - a. Sends an update notification to all of its DCR Slaves.
 - b. Sends “New Device Added” events to all applications using the Master.
4. Each DCR Slave receiving update notification from its Master then:
 - Synchronizes with its Master to get the device details.
 - Sends “New Device Added” events to all applications using that DCR Slave.

Users whose application resides on the same CWCS Server with a Slave can use the application UI or the DCR local API to add a new device. When this happens:

1. The Slave calls the Master to add the device to the Master’s Repository.
2. If the DCR Master is down, the new device is a duplicate, or there is another problem, the Master returns an error to the Slave. The Slave returns an error to the application, and no updates take place on the Master or the Slave.
3. If there are no problems, the DCR Master adds the device to its repository.
4. The DCR Master then:
 - a. Sends an update notification to its DCR Slaves.
 - b. Sends “New Device Added” events to the applications using that DCR Master.
5. Each DCR Slave receiving update notification from its Master then:
 - Synchronizes with its Master to get the device details.
 - Sends “New Device Added” events to all applications using that DCR Slave.

How DCR Modifies Devices

Users whose application resides on the same CWCS server with a DCR Master can use the local DCR APIs or the DCA GUI to update devices on that Master. When this happens::

1. The DCR Master checks whether the modifications will turn the modified device into a duplicate of another device in the repository. If it is a duplicate, the operation fails.
2. If the device update is not a duplicate, the DCR Master updates the device in the repository.
3. The DCR Master then:
 - a. Sends an update notification to its DCR Slaves.
 - b. Sends “Device Updated” events to all applications using the Master.
4. Each DCR Slave receiving the update notification from its Master then:
 - Synchronizes with its Master to get the device details.
 - Sends “Device Updated” events to all applications using that DCR Slave.

CISCO CONFIDENTIAL

Users whose application resides on the same CWCS Server with a Slave can use the application UI or the DCR local API to update an existing device. When this happens:

1. The Slave calls the Master to modify the device in the Master's Repository.
2. If the DCR Master is down, the updated device is a duplicate, or there is another problem, the Master returns an error to the Slave. The Slave returns an error to the application, and no updates take place on the Master or the Slave.
3. If there are no problems, the DCR Master adds the device to its repository.
4. The DCR Master then:
 - a. Sends an update notification to its DCR Slaves.
 - b. Sends "Device Updated" events to all applications using the Master.
5. Each DCR Slave receiving update notification from its Master will:
 - Synchronize with its Master and get the device details.
 - Send "Device Updated" events to all applications using that DCR Slave.

How DCR Deletes Devices

Applications cannot delete devices directly. Only administrators using the DCA GUI or the DCR CLI on a DCR Server in Master or Standalone mode can delete devices. When this happens, the Master:

1. Deletes the device from the Master repository.
2. Sends "Device Deleted" notifications to its DCR Slaves .
3. Sends "Device Deleted" events to all applications using that DCR Master.
4. Each DCR Slave receiving the "Device Deleted" notifications from its Master will:
 - Synchronize with its Master.
 - Send "Device Deleted" events to all applications using that DCR Slave.

How DCR Secures Device and Credentials Data

DCR allows access to device and credentials data only by authenticated, authorized users. DCR provides a basic level of security by storing device credentials as encrypted data (other attribute data is in plain text), and by using secure communications protocols.

DCR also uses the following security keys:

- Username: The user name of the accessing user.
- Password: The accessing user's password.
- Secret Key: A password supplied by the administrator during CWCS installation, or specified using the **Security > Server >System Identity Setup** option on the CiscoWorks Home Page. Note that there is also a Secret User associated with this Secret Key.

How DCR uses these security keys varies with the kind of communication taking place:

- **Master Notifications to Slaves:** These notifications receive no extra security. A DCR Master calls Slave APIs only to notify its Slaves that changes have occurred in its repository. The notifications do not contain credentials data or attributes other than the internal DCR Device IDs. Slaves must call the Master to get the updated attribute or credentials data.

CISCO CONFIDENTIAL

- **Slave Calls to Master:** These occur during synchronization, in response to a notification from a Master, and when applications use a Slave to add or update devices and credentials. All such Slave requests must pass either the Secret Key or Password to authenticate, and the User Name to authorize, access to the Master. Note that third-party applications calling a DCR Master remotely via northbound API may pass the Secret Key only.
- **DCR Events to Applications:** These events receive no extra security. DCR sends events to applications only to notify them that devices or credentials data in its repository has been updated. The events do not contain credentials, or anything but basic device identifiers. Applications must call DCR to receive the updated device or credentials data.
- **Application Calls to DCR:** Applications on the same server as the called DCR use the DCR local APIs. DCR assumes the client was successfully authenticated before the call, and requires only the user name to authorize access. Applications residing on a remote server can call DCR via northbound APIs, but must provide the user name and password. DCR authenticates and authorizes using the user name and password.

About DCR Data Storage

The following topics explain how DCR stores device lists, their attributes and credentials:

- [About the DCR Device ID](#)
- [How DCR Stores Attributes](#)
- [How DCR Stores Credentials](#)
- [How DCR Stores Proxy Device Data](#)
- [How DCR Stores Enable-Mode Passwords](#)
- [About User-Defined Fields](#)



Note

DCR and its APIs do not communicate with devices directly. The code that interacts directly with devices and fetches their data for storage in DCR is the responsibility of the application.

About the DCR Device ID

For every device, DCR maintains an internally generated, unique, sequential number called the DCR Device ID. This ID identifies the device's record in the DCR database. Your application must use this ID when communicating with DCR about a specific device and its data. DCR does not re-use deleted DCR Device IDs. Whenever an administrator deletes a DCR device using the DCA GUI or the DCR CLI, all information for that device, including the DCR Device ID, is removed from DCR.

Be sure not to confuse the DCR Device ID with other attributes used to identify the device in the network, such as the `host_name` or `sysObjectID` (for a complete list of these other attributes, see the [“How DCR Stores Attributes”](#) section on page 14-10).

If your application maintains its own set of device identifiers, you will need to create and maintain tables that map your application's device identifiers to the DCR Device IDs (see the [“Guidelines for DCR Application Development”](#) section on page 14-38).

CISCO CONFIDENTIAL**How DCR Stores Attributes**

In DCR, an attribute is all device data other than credentials (see the “[How DCR Stores Credentials](#)” section on page 14-10). Attribute data consists of displayable strings of printable ASCII characters. Attributes are considered unique to each device (e.g., `host_name` and `management_ip_address`).

DCR stores values for the attributes shown in [Table 14-1](#) for all standard devices. Some proxy devices store additional attributes (see the “[Integrating DCR with OGS](#)” section on page 14-13).

Table 14-1 DCR Device Standard Attributes

| Attribute | Description |
|---|---|
| <code>host_name</code> | Device host name. |
| <code>domain_name</code> | Domain name of the device. |
| <code>management_ip_address</code> | The IP address used to access the device. Both IPv4 and IPv6 address types are supported. Required if <code>host_name</code> is not specified. |
| <code>display_name</code> | The name the user wants the device to have in reports or graphical displays. This value can be derived from <code>host_name</code> or <code>management_ip_address</code> . Required. |
| <code>sysObjectID</code> ¹ | A string with the <code>sysObjectID</code> value. This attribute is required, but DCR will fill in the value automatically if the <code>mdf_type</code> is specified. |
| <code>mdf_type</code> ¹ | A normative name for the device type, taken from Cisco's Meta Data Framework (MDF) database. This attribute is required, but DCR will fill in the value automatically if the <code>sysObjectID</code> is specified. |
| <code>dcr_device_type</code> ² | A name for the device type to be used in DCR. It has meaning only in the DCR context. This is not the actual network device attribute. |

1. You cannot leave both of these attributes blank, but you can specify either one or both of them as “unknown”. DCR will determine the `sysObjectID` value automatically if only the `mdf_type` is specified, and vice versa. If you enter both, DCR will *not* check that they map properly.
2. You must specify a value for `dcr_device_type` when adding a new device. The `dcr_device_type` and DCR Device Category are different terms for the same construct (e.g., the value of a device's `dcr_device_type` is set when you call the `SetDCRDeviceCategory` method on that device).

**Note**

If for any of the attributes, the value is modified to an empty string, the attribute itself will not be disassociated from the corresponding device. Instead, the attribute will be stored with an empty value. If a device is added in DCR without an IP address, then DCR returns null for the attribute, `management_ip_address`. But if IP address is removed (ie, set to empty) from an existing DCR device, then DCR returns empty string.

How DCR Stores Credentials

DCR credentials are the attribute values that applications use to access and operate on devices. For example, if the device has SNMP enabled, the SNMP read-only and read-write community strings used to access that device are considered DCR credentials. All credentials are encrypted and not displayable in encrypted form.

DCR associates the credentials shown in [Table 14-2](#) with all standard devices, DSBU Clusters and DSBU Cluster Members. Some proxy devices use a different mix of credentials (see the “[How DCR Stores Proxy Device Data](#)” section on page 14-11).

CISCO CONFIDENTIAL**Table 14-2 DCR Device Standard Credentials**

| Credential | Description |
|-------------------------|---|
| primary_username | The primary user name used to access the device. |
| primary_password | The password for the primary_username. |
| primary_enable_password | The device's primary "enable password" or "enable secret" password. See the "How DCR Stores Enable-Mode Passwords" section on page 14-13. |
| snmp_v2_ro_comm_string | The device's SNMP V2 read-only community string. |
| snmp_v2_rw_comm_string | The device's SNMP V2 read/write community string. |
| snmp_v3_user_id | The device's SNMP V3 user ID. |
| snmp_v3_password | The device's SNMP V3 password. |
| snmp_v3_engine_ID | The device's SNMP V3 engine ID. |
| snmp_v3_auth_algorithm | The SNMP V3 authorization algorithm used on the device (i.e., MD5 or SHA-1). |
| rxboot_mode_username | The device's diagnostic user ID. |
| rxboot_mode_password | The device's diagnostic password. |

The following credentials and attributes are added as part of Common Services 3.0 Service Pack 1 (CS3.0 SP1)

Table 14-3 Credential and Attributes Added in CS 3.0 SP1

| Credential/Attribute | Description |
|----------------------|---|
| http_username | HTTP Username |
| http_password | HTTP password |
| http_port | HTTP Port |
| https_port | HTTPS Port |
| http_mode | Current transfer mode (http or https) |
| cert_common_name | Common name attribute value in the server's certificate |

How DCR Stores Proxy Device Data

For standard network devices, DCR stores the attributes and credentials listed in the ["How DCR Stores Attributes"](#) section on page 14-10 and in the ["How DCR Stores Credentials"](#) section on page 14-10. Some proxy devices use different attribute and credential schemes. DCR supports these devices with attributes and credentials appropriate to them, as follows:

- **DSBU Clusters:** Each DSBU cluster has its own database record, which represents a "logical" DSBU cluster. This record does not point to any one physical device in the cluster, and stores the credentials of the DSBU Commander. For DSBU Clusters, DCR stores the attributes and credentials used for standard devices.

CISCO CONFIDENTIAL

- **DSBU Cluster Members:** Each DSBU Cluster Member has its own host_name, sysObjectID, and mdf_type value, and uses the same credentials as the DSBU Cluster. For DSBU Cluster Members, DCR stores the same attributes and credentials as for standard devices, plus the credentials shown in [Table 14-4](#).

Table 14-4 Additional DSBU Cluster Member Credentials

| Credential | Description |
|--------------------|---|
| dsbu_member_number | The number of the DSBU Cluster member. This number represents the order in which the device was added to the cluster. |
| parent_dsbu_id | The DCR Device ID of the parent DSBU Cluster device. |

- **AUS Devices:** For the AUS device itself, DCR stores the same attribute data used for standard devices, but does not use standard credentials. Instead, it stores the credentials shown in [Table 14-5](#).

Table 14-5 AUS-Specific Credentials

| Credential | Description |
|--------------|---|
| aus_url | The URL for the AUS device. |
| aus_port | The port number of the AUS service running on the AUS device. |
| aus_username | The user login providing access to the AUS device. |
| aus_password | The password for the corresponding aus_username. |

- **AUS- Managed Devices:** For each device managed by an AUS device, DCR stores the same attributes as for standard devices, plus the additional device_identity attribute. It does not use standard credentials; instead, it stores the credentials shown in [Table 14-6](#).

Table 14-6 AUS-Managed Device: Additional Attribute and Device-Specific Credentials

| Attribute/Credential | Description |
|----------------------|--|
| device_identity | This attribute is a string uniquely identifying the AUS-managed device. |
| aus_username | The user login providing access to the AUS-managed device. |
| aus_password | The password for the corresponding aus_username. |
| parent_aus_id | The DCR Device ID of the managing AUS device. |

The following are supported from CS3.0 SP1.

- **CNS Configuration Engine(CNS Server):**
For the CNS Server, DCR stores the same attribute data used for standard devices.
- **CNS Managed Devices:**
For the CNS Server, DCR stores the same attribute data used for standard devices. In addition to this, it will have the parent_cns_id attribute

CISCO CONFIDENTIAL**Table 14-7** CNS Managed Devices Specific Device Credentials

| Attribute/Credential | Description |
|----------------------|---|
| parent_cns_id | Device ID of the parent CNS server (CNS Configuration Engine) |
| cns_config_id | CNS Config ID of the device. |
| cns_event_id | CNS Event ID of the device. |
| cns_image_id | CNS Image ID of the device. |

How DCR Stores Enable-Mode Passwords

Most managed devices feature “enable password” and “enable secret” commands. Both permit an Enable Mode password to be set and stored within the configuration for a managed device. Only administrators who know this password can change the device configuration. The “enable secret” command adds an extra layer of protection by encrypting the stored password, preventing anyone from discovering the password by examining the device configuration or a copy of it.

DCR retains only one Enable Mode password. If the Enable Mode password was set using the “enable password” command only, then DCR will store only that password. If an “enable secret” command has been used as well as (or instead of) the “enable password” command, the “enable secret” password will take precedence.

If you are importing device data from a product that does not use DCR, you may find a user database where both the “enable password” and “enable secret” have been set. In this case, the “enable password” should be discarded, and the “enable secret” password moved into the new DCR database. If only one of the values is populated in the source database, then only that value should be migrated.

About User-Defined Fields

The DCR database allows users to specify up to 10 user-defined fields. These fields, once specified by a user, become part of the device record structure and their values are stored as part of device records. While application developers can access data in user-defined fields for reporting and other purposes, they cannot add or change user-defined fields programmatically. User-defined fields are for users only, and must be added using the DCA GUI only.

Integrating DCR with OGS

To enable operations on groups of devices, DCR provides the following pre-defined groups:

- System Defined: Includes all devices, arranged by mdm_type value.
- User Defined: Includes all devices, arranged by user-specified attribute values.

OGS (see [Chapter 30, “Using Object Grouping Services”](#)) incorporates these DCR Groups and provides the underlying shared and secured device grouping services to DCR.

The Device Selector provided with the CiscoWorks Home Page (see [Chapter 7, “Using the CiscoWorks Home Page”](#)) performs basic filtering of devices based on IP address, host name, display name, user-defined attributes, etc. Once the DCR filtering is done, the DCR Device Selector will show the filtered device list within the predefined DCR Groups provided by OGS.

CISCO CONFIDENTIAL**Integrating DCR with ACS**

When DCR starts and the CWCS Server is in ACS mode, DCR will register all known devices in its repository with CAM. This registration process enables DCR APIs to authorize the devices for the user appropriately. DCR also registers devices with CAM whenever device(s) are added, as part of the “add” operation, and updates CAM appropriately whenever DCR devices are modified.

Each DCR device is mapped to an ACS device based on the device’s IP address in both DCR and ACS. If a DCR device has no IP address, then its DCR display name is mapped to an ACS host name.

The DCR APIs authorize the user at the task level at all times, using the DCR and ACS Task IDs shown in [Table 14-8](#) (this table ignores DCR-related ACS tasks that cannot be triggered by DCR API calls). See [Table 14-9](#) for the list of default DCR Task-to-Role mappings; this table includes some DCR-related ACS tasks that can only be triggered using the DCR GUI.

Device-level authorizations are applied only when the CWCS Server is in ACS mode. Note that an application running on the same server as the CWCS Server can disable task- and device-level authorization for local DCR access. If your application uses the APIs to perform DCR tasks, and authorizes their use by particular users based on role, be sure that your application-level authorizations match those used by the DCR API tasks.

Table 14-8 DCR APIs and Associated Tasks

| DCR API | DCR Task | ACS Task |
|--|--|--------------|
| addDevice, addDevices | ADD_DEVICE_TASK | Add |
| getDCRDomainID | None | None |
| getDCRID | None | None |
| getDCRUpdates | VIEW_CREDENTIAL_TASK | View |
| getDeletedDevices(DeviceId[] managedDeviceids, APIExtraInfo apiExtraInfo) throws DCRException // all devices | VIEW_CREDENTIAL_TASK | View |
| public synchronized Device[] getDeletedDevices(long transactionId, APIExtraInfo apiExtraInfo) // all devices | VIEW_CREDENTIAL_TASK
Deprecated. Do not use this API. | View |
| getDevice, getDevices | VIEW_CREDENTIAL_TASK | View |
| getDeviceIdentifiers | VIEW_DEVICE_TASK | View Devices |
| getDevices(DeviceId [] id, APIExtraInfo apiExtraInfo) | VIEW_CREDENTIAL_TASK | View |
| getDevices(DeviceId[] id, String[] attList, APIExtraInfo apiExtraInfo) | VIEW_CREDENTIAL_TASK | View |
| Device[] getDevices(long timeStamp, APIExtraInfo apiExtraInfo) throws DCRException // all devices | VIEW_CREDENTIAL_TASK | View |
| getIdentityAttributes | VIEW_DEVICE_TASK | View Devices |
| getMasterDCRID | None | None |
| getMatchingDevices | VIEW_DEVICE_TASK | View Devices |
| getMaxTransactionID | None | None |

CISCO CONFIDENTIAL**Table 14-8 DCR APIs and Associated Tasks (continued)**

| DCR API | DCR Task | ACS Task |
|-----------------------------|----------------------|--|
| getMissingDCRDevicesinACS | VIEW_DEVICE_TASK | View Devices |
| getNewDevices | VIEW_CREDENTIAL_TASK | View |
| getUnauthorizedDevices | VIEW_DEVICE_TASK | View Devices |
| getUpdatedDevices | VIEW_CREDENTIAL_TASK | View |
| isMasterDCR | None | None |
| isMasterDCRRunning | None | None |
| isRunning | None | None |
| registerForHTTPEvents | REG_APP_TASK | Register/Unregister 3rd Party Application in DCR |
| unregisterForHTTPEvents | REG_APP_TASK | Register/Unregister 3rd Party Application in DCR |
| updateDevice, updateDevices | UPDATE_DEVICE_TASK | Edit |

Table 14-9 DCR Task-to-Role Mapping

| Task/Role | Help Desk | Approver | Network Operator | Network Administrator | System Administrator |
|-------------------------------------|-----------|----------|------------------|-----------------------|----------------------|
| Add | | | X | X | X |
| Edit | | | | X | X |
| Delete | | | | X | X |
| Reports | X | X | X | X | X |
| View | | X | | X | X |
| View Devices | X | X | X | X | X |
| Change Mode | | | | X | X |
| Add User Defined Fields in DCR | | | | X | X |
| Modify User Defined Fields in DCR | | | | X | X |
| Delete User Defined Fields from DCR | | | | X | X |
| Export | | | | X | X |
| Bulk Import | | | | X | X |

About DCR Events

DCR uses CWCS Event Services Software (ESS) (see [Chapter 19, “Using Event Services Software”](#)) to broadcast events to all applications on the same CWCS Server. DCR clients listening to these events can choose to filter out unwanted data or events of no interest (e.g., events for classes of devices the application does not support). When filtering data, applications can query DCR for additional device details using the DCR APIs.

CISCO CONFIDENTIAL

Applications that cannot use ESS can receive DCR events via HTTP. An application wanting to use this alternative method must first use the DCR API to register its HTTP URL with DCR. DCR will then use this URL to send events to the non-ESS application.

The DCR Server never listens to events broadcast by applications.

DCR generates:

- **Device events** whenever there is a change in the attributes or credentials of devices stored in the DCR repository (e.g. device additions, imports, etc.) For details, see the [“About DCR Device Events” section on page 14-17](#).
- **Process events** when the DCR Server starts, stops, or changes modes. For details, see the [“About DCR Process Events” section on page 14-19](#).
- **Restore events** whenever DCR data is restored from backup. For details, see the [“About DCR Restore Events” section on page 14-21](#).

The normal process of database backup and restore can also generate combinations of these events and accompanying notifications. For a summary of these, see the [“About DCR Events During Backup and Restore” section on page 14-22](#).

In Master/Slave implementations, the DCR Notifier generates synchronization and update notifications, so DCR Slaves can get device updates from Masters. These notifications are not broadcast to DCR client applications. The events include all significant repository changes, such as device additions, deletions, credential updates, etc.

Note that events are not the same as notifications sent from a DCR Master to a Slave (see the [“About the DCR Components” section on page 14-5](#)). With DCR Servers in Standalone mode, events are generated directly and sent to applications who use that DCR. When a DCR Master’s devices are updated, it generates events directly to applications using it, and then sends notifications to its Slaves. These Slaves, in turn, get their updates from the Master, and then generate events for the applications using the Slaves.

The details of event generation for DCR Slaves are the same as in DCR Master. From the application's perspective:

- It will receive events from DCR when there is an update in DCR. The events can be for new, existing, or deleted devices in DCR.
- The application can fetch device lists and credentials data from DCR as needed.

When a new application (that does not contain any device credential data) is installed, the application can get the device list available from DCR and present the devices in a pick list. The user can then select the devices from the list for management.

About the DCR Domain ID and Transaction ID

The Master DCR Server in every Master/Slave DCR setup has its own DCR Domain ID. The DCR Domain ID identifies the logical DCR domain in which the Master and its Slaves participate. Your application must:

- Maintain the DCR Domain ID as part of its data.
- Check to see if the DCR Domain ID has changed:
 - At application startup. You can do this by calling `getDCRDomainID()` during startup.
 - Whenever the application receives a `DCR_DATA_RESTORED` or `DCR_DATA_RESTORED_FROM_DIFFERENT_DOMAIN` event. You can do this by comparing the DCR Domain ID contained in the event with the DCR Domain ID stored in the application database.

CISCO CONFIDENTIAL

- If the DCR Domain ID has changed with a DCR_DATA_RESTORED event, update the DCR Domain ID in the application database.
- If the DCR Domain ID has changed with a DCR_DATA_RESTORED_FROM_DIFFERENT_DOMAIN event:
 - Update the DCR Domain ID in the application database.
 - Clean up and refresh the device list.

Every DCR Server update of any kind (e.g., device additions, deletions, updates, etc.) has a transaction ID, which is simply a serial number. The DCR Transaction ID is the ID of the last transaction (e.g., device update, deletion, addition, etc.) an application conducted with its DCR Server. This transaction ID is maintained in the application database, and allows the application to determine whether updates occurred while the application was offline. To make use of it, your application must

- Maintain the DCR Transaction ID in its own database.
- At application startup, check to see if the last (or maximum) transaction ID recorded on the DCR Server is greater than the DCR Transaction ID recorded in the application database. You can get the last transaction ID for the Server by calling `getMaxDcrTransactionId()` during startup.
- If the transaction ID on the server is higher than the DCR Transaction ID stored in the application:
 - Update the DCR Transaction ID in the application database.
 - Get all the device updates since the last transaction recorded in the application.

For examples of code making use of the Domain ID and the Transaction ID, see the [“Using DCR Domain and Transaction IDs”](#) section on page 14-44.

About DCR Device Events

In addition to events for DCR changes (see the [“About DCR Process Events”](#) section on page 14-19) and data restores (see the [“About DCR Restore Events”](#) section on page 14-21), DCR broadcasts events whenever devices are updated (including addition or deletion).

A DCR device event contains the event’s:

- Subject (also treated as the event ID). For example: `Event Subject Name = "cisco.mgmt.cw.cmf.dcr";`
- Data, which consists of:
 - The type of device change. For example: `DEVICE_ADDED`.
 - Identifying details for the device(s) affected by the event. For example: the Device ID, `sysObjectID`, etc.
 - The event Timestamp.

[Table 14-10](#) shows the data for all types of device events. Applications can use this data to decide whether to process the event or not, and to query DCR for details about the target device(s).

CISCO CONFIDENTIAL**Table 14-10 DCR Device Update Event Data**

| Device Change | Event Data |
|---------------|--|
| Added | <pre> <DCREvent> <EventType>DEVICES_ADDED</EventType> <EventSource> <AppName>DCRUI</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> <Devices> <Device> <DeviceId>14</DeviceId> <SysObjectId>222</SysObjectId> <IpAddress>sds</IpAddress> <HostName>>null</HostName> <DisplayName>sd44s</DisplayName> <TransactionId>556598</TransactionId> <MDFID>12345</MDFID> </Device> </Devices> </DCREvent> </pre> |
| Added in Bulk | <pre> <DCREvent> <EventType>BULK_DEVICES_ADDED</EventType> <EventSource> <AppName>DCRUI</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent> </pre> |
| Deleted | <pre> <DCREvent> <EventType>DEVICES_DELETED</EventType> <EventSource> <AppName>DCRUI</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> <Devices> <Device> <DeviceId>14</DeviceId> <SysObjectId>222</SysObjectId> <IpAddress>sds</IpAddress> <HostName>>null</HostName> <DisplayName>sd44s</DisplayName> <TransactionId>>null</TransactionId> <MDFID>12345</MDFID> </Device> </Devices> </DCREvent> </pre> |

CISCO CONFIDENTIAL**Table 14-10 DCR Device Update Event Data (continued)**

| Device Change | Event Data |
|-----------------|--|
| Deleted in Bulk | <pre> <DCREvent> <EventType>BULK_DEVICES_DELETED</EventType> <EventSource> <AppName>DCRUI</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent> </pre> |
| Updated | <pre> <DCREvent> <EventType>DEVICES_UPDATED</EventType> <EventSource> <AppName>DCRUI</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> <Devices> <Device> <DeviceId>14</DeviceId> <SysObjectId>222</SysObjectId> <IpAddress>sds</IpAddress> <HostName>>null</HostName> <DisplayName>sd44s</DisplayName> <TransactionId>>null</TransactionId> <MDFID>12345</MDFID> </Device> </Devices> </DCREvent> </pre> |
| Updated in Bulk | <pre> <DCREvent> <EventType>BULK_DEVICES_UPDATED</EventType> <EventSource> <AppName>DCRUI</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent> </pre> |

About DCR Process Events

DCR broadcasts process events whenever the DCR Server starts, which is a common event. It also broadcasts events when the server changes modes, which happens much less often. A DCR process event consists of the event's:

- Subject (also treated as the event ID).
For example: Event Subject Name = "cisco.mgmt.cw.cmf.dcr";.
- Data, which consists of:
 - The type of change. For example: MASTER_CHANGED_TO_SLAVE.
 - Identifying details for the DCR affected by the event. For example: "DCR Server 1".

CISCO CONFIDENTIAL

Example 14-1 shows the event data for the only normal process event, a DCR Server start. Table 14-11 shows event data for the uncommon DCR mode-change process events. All of these events are broadcast because applications (such as OGS Server) need to know about them. Most applications can ignore process events *unless* they are also accompanied by a “data restored” event, in which case applications *must* receive and process them (see the “About DCR Restore Events” section on page 14-21).

Example 14-1 DCR Server Start Event Data (continued)

```
<DCREvent>
  <EventType>DCR_SERVER_START</EventType>
  <EventSource>
    <AppName>DCR Server</AppName>
    <AppVersion>1.0</AppVersion>
    <AppHostName>server1</AppHostName>
  </EventSource>
</DCREvent>
```

Table 14-11 DCR Mode-Change Event Data

Mode Change	Event Data
Standalone to Master	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>STANDALONE_CHANGED_TO_MASTER</EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>
Standalone to Slave	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType> STANDALONE_CHANGED_TO_SLAVE</EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>
Slave to Master	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType> SLAVE_CHANGED_TO_MASTER</EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>

CISCO CONFIDENTIAL**Table 14-11 DCR Mode-Change Event Data (continued)**

Mode Change	Event Data
Slave to Standalone	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>SLAVE_CHANGED_TO_STANDALONE</EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>
Master to Slave	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>MASTER_CHANGED_TO_SLAVE </EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>
Master to Standalone	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>MASTER_CHANGED_TO_STANDALONE </EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>
Slave to New Master	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>SLAVE_CHANGED_TO_NEW_MASTER</EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>

About DCR Restore Events

DCR sends the restore events shown in [Table 14-12](#) when DCR data is changed due to specific DCR mode changes or a data restore from backup. The restore events are sent in different conditions:

- DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN: DCR will send this event when:
 - A DCR Server in Standalone or Master mode changes to Slave mode. The new Slave-mode DCR Server receives this event before it begins receiving DCR DATA RESTORED events.
 - A DCR Server in Slave mode is reassigned to a DCR Master in a different domain.
 - A DCR Server in Slave mode has its DCR Master change its host name.
 - A CWCS restore is performed and the restore contains different domain data.

Applications receiving this event must:

- Alert the application administrator to act appropriately.
- Clean up the application-managed device list.
- Call `getDCRUpdates()` to refresh the application device list.

CISCO CONFIDENTIAL

- DCR DATA RESTORED: DCR will send this event when:
 - A DCR Server in Standalone or Master mode changes to Slave mode. The new Slave-mode DCR Server receives this event after the DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event, and continues to receive it until the Slave-mode Server syncs with its new Master.
 - When a DCR Server in Master mode changes its port and the user re-registers with a Slave with this Master.
 - When a DCR Server in Slave mode changes its port and the user re-registers with this Slave with its Master.
 - When DCR Server in Slave mode changes to Standalone or Master mode.

Applications receiving this event are expected to call getDCRUpdates() to get all new, updated, or deleted devices from DCR.

Table 14-12 DCR Restore Events

Restore Event	Event Data
Data restored from different domain	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>DCR_DATA_RESTORED_FROM_DIFFERENT_DCR_DOMAIN </EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>
Data restored from same domain	<pre><DCREvent> <DCRDomainID>Group123456</DCRDomainID> <EventType>DCR_DATA_RESTORED</EventType> <EventSource> <AppName>DCR Server</AppName> <AppVersion>1.0</AppVersion> <AppHostName>server1</AppHostName> </EventSource> </DCREvent></pre>

About DCR Events During Backup and Restore

DCR uses the CWCS backup and restore framework (see the [Chapter 12, “Using Backup and Restore”](#)) to protect the DCR data stored in the main CWCS database and the DCR configuration file.

Data changes are a normal part of any restore from a backup. However, because DCR is a distributed system with varying modes, it is also possible for any restored DCR to:

- Change modes (see the [“How DCR Masters and Slaves Interact”](#) section on page 14-6). For example, a Standalone DCR can be set after a backup to act as a Slave. When the restore is performed, it will be reset to Standalone mode.
- Change master/slave relationships. For example: A DCR Slave may be using Master A at the time a backup is taken. Later, the domain will be changed to use Master B, and the Slave reset to use Master B. When the restore is performed, the Slave will attempt to use Master A.

CISCO CONFIDENTIAL

- Change management domains. For example: A DCR Slave using a Master in domain A can be reset to become a Slave of a Master in domain B. When the restore is performed, the Slave will attempt to replicate with the Master in domain A.

Because any restore can involve a combination of data, mode, master/slave and domain changes, DCR provides special processing functions to handle:

- Domain Changes: DCRs running in Master or Slave mode always have an associated DCR Group ID that indicates the Server's management domain. This Group ID is generated when a DCR is set to Master mode, and communicated to all Slaves later assigned to that Master. The restore process checks the DCR Group ID for appropriate action. If the Group ID changes due to the restore, then DCR broadcasts a "DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN" event. If the DCR Group ID does not change, DCR assumes that no domain change took place, and broadcasts a "DCR DATA RESTORED" event.
- Mode and Master/Slave Changes: These vary according to the individual DCR's working or pre-restore mode (the mode at the time the restore process is started), its end or post-restore mode (the mode after the restore is finished), and its existing Master/Slave relationships. Mode changes for DCR Servers in Master mode also generate "DCR MASTER MODE CHANGE" events. [Table 14-13](#) summarizes all possible mode and master/slave changes for restored DCRs and the actions they take in response.

Applications will need to process domain, mode, and master/slave change notifications from DCRs appropriately, as well as processing data change events. This includes:

- Synchronizing with the DCR during application startup.
- Making provision for the fact that, during a restore, the application may be down for a significant period and may not have a chance to process restore-related events.
- Querying DCR, using DCR Device IDs, for information about deleted and updated devices. Applications that persist any device identity attributes must also pass them to DCR so that DCR can identify whether the device information is modified. DCRProxy provides several APIs to detect deleted and updated devices, including `getDeletedDevices()` and `getDCRUpdates()` (see the "[How DCR Masters and Slaves Interact](#)" section on page 14-6).

CISCO CONFIDENTIAL**Table 14-13 DCR Restore Behavior**

If the Post-Restore (End) Mode is:	And the Pre-Restore (Working) Mode was:		
	Standalone	Slave	Master
Standalone	<p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>	<p>After restore, the former Slave unregisters from the old Master and deletes old Master information. If the former Slave could not unregister and the old Master sends it notifications, it ignores them (as it is now in Standalone mode).</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>	<p>After restore, the former Master sends DCR MASTER MODE CHANGED events to all its former Slaves. These Slaves delete the Master contact information.</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>
Slave	<p>After restore, the server is in Master mode. We do this to preserve the device context¹. To finish conversion to Slave mode, use the GUI or CLI commands.</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>	<p>After restore, if the DCR Group ID is different, then the server moves to Standalone mode, and sends a mode change event and a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event.</p> <p>If the DCR Group ID is same and Master is different, then the DCR will unregister with the old Master, delete all data it took from the old Master, register and sync with the new Master, and send a “DCR DATA RESTORED” event. Applications must process these events and query DCR to get updates.</p>	<p>After restore, the server is in Slave mode. It sends a DCR MASTER MODE CHANGED event to all former Slaves. These Slaves delete the contact information for the former Master.</p> <p>The new DCR Slave uses the restored Slave configuration information to contact, re-register and sync data with its Master.</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>

CISCO CONFIDENTIAL**Table 14-13 DCR Restore Behavior (continued)**

If the Post-Restore (End) Mode is:	And the Pre-Restore (Working) Mode was:		
	Standalone	Slave	Master
Master	<p>After restore, the server is in Master mode. Some Slave information may be invalid², so the new DCR Master will delete all Slave information.</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>	<p>After restore, the server is in Master mode. Some Slave information may be invalid², so the new Master will delete all Slave information. It also deletes all Master contact information left over from Slave mode.</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>	<p>After restore, the server is still in Master mode. It compares information about its Slaves that comes from the backup with the Slave information from pre-restore operations. It retains Slaves that exist in both and deletes all others.</p> <p>If the DCR Group ID is different, DCR sends a DCR DATA RESTORED FROM DIFFERENT DCR DOMAIN event. If the DCR Group ID is the same, DCR sends a DCR DATA RESTORED event. Applications must process these events and query DCR to get updates.</p>

1. Suppose we have a DCR Server in Standalone mode, and restore it using Slave data, If we convert directly to Slave mode (retain the Slave configuration from the backup) then the context of the devices known to the applications using the Standalone server will be different from the devices in the Slave. Since Slave and Master data are tightly coupled, the risk is high that all the restored Slave data will be overwritten or lost as soon as communications with the Master begin.
2. Some of the Slaves included in the backup information for this Master may have moved to another Master.

Using DCR

The following topics provide guidelines on and examples of how to integrate DCR with your application:

- [Getting Started With DCR](#)
- [DCR Tasks to Perform During Application Startup](#)
- [Using the DCR APIs](#)
- [Responding to DCR Events](#)
- [Using DCR Domain and Transaction IDs](#)
- [Using the DCR Command-Line Interface](#)
- [Enhancing DCR Performance](#)

CISCO CONFIDENTIAL**Getting Started With DCR**

Use the following steps to create and populate a standalone DCR Server.

Step 1 Install CiscoWorks Common Services (CWCS).

DCR Server is included in this version of CWCS, and will be installed automatically. Upon conclusion of the installation, the CWCS Daemon Manager will start a local DCR Server. The DCR Server will run in Standalone mode, so it will not be part of a management domain or share devices with any other DCR.

If you need to manually control the DCR Server at any time after installation, enter the following at the command line:

- To stop DCR, enter `pdterm DCRServer`
- To start DCR, enter `pdexec DCRServer`

Step 2 Log on to CWCS.

Point your browser to the CWCS installation at `http://server_ip_address:1741`. Log into CiscoWorks by providing a username and password (the default for both is `admin`).

Step 3 Start populating the local DCR Server by manually entering data for a few devices, as follows:

- a. On the Common Services application panel, select **Device & Credentials Admin > Device Management**. CiscoWorks displays the Device Management screen.
- b. Click **Add** to begin adding devices. CiscoWorks displays a Device Properties page.
- c. Enter the required device Display Name and select the Device Type.
- d. Enter one of the following required attributes: IP address, Host Name, or AUS Device ID (only if the device is AUS-managed).
- e. Click **Add to list** to add the device to DCR.
- f. Repeat steps a through e for the other devices you want to add to DCR.
- g. When you have added all the devices you want, click **Next** to specify credentials for them.
- h. When you have added all the credentials needed, click **Next** to specify user-defined fields.
- i. When you are finished, click **Finish**.

Step 4 If you want a large number of devices, you can use the DCR Export feature to create a CSV file, and edit the CSV file to add all the other devices you want. Then use the Import feature to load the updated CSV file. For example:

- a. On the Device Management screen, click **Export** to export to a CSV file the devices you have already added.
- b. Select **File**, specify the file name, and then click **Export**.
- c. When the export operation is completed, edit the CSV file as needed. You can use an ASCII text editor to do this.

Alternatively, you can use a spreadsheet application to create a CSV file from scratch, using the exported CSV file as a model for your entries. You can then load it using the following steps.

- d. On the Device Management screen, click **Import**.
- e. Select **File**, specify the file name of the CSV file you edited, and then click **Import**.

CISCO CONFIDENTIAL

DCR Tasks to Perform During Application Startup

Every application using DCR should be sure to complete the following tasks at startup:

**Note**

After start-up, your application should not process DCR events sent before the DCR_SERVER_START event. Ideally, your application should set a Daemon Manager dependency on DCRServer to avoid this.

1. Get the current DCR transaction ID stored in the application database.
2. If the transaction ID is zero, then this is the first time the application has been started. In this case:
 - a. Call the `getNewDevices()` API and get the device list from DCR.
 - b. Update the application-managed device list from the DCR device list.
 - c. Get the DCR Domain ID and store it in the application database.
 - d. Derive the maximum transaction ID from the device list and update it in the application database.
3. If the transaction ID is not zero, fetch the current DCR Domain ID from the DCR Server and compare it with the DCR Domain ID stored in the application database.
4. If the Server and application DCR Domain IDs are different:
 - a. If necessary, clean the application-managed device list, or perform any other required application-specific action.
 - b. Set the application transaction ID to zero.
 - c. Call the `getNewDevices()` API and get the device list from DCR.
 - d. Update the application-managed device list from the DCR device list.
5. If the Server and application DCR Domain IDs are identical:
 - a. Call the `getDCRUpdates()` API and get the device updates from DCR
 - b. Update the application-managed devices from the DCR device updates.

For an example of startup code that performs all of these tasks, see the [“Using DCR Domain and Transaction IDs”](#) section on page 14-44.

Using the DCR APIs

Once you have set up a working DCR Server as described in the [“Getting Started With DCR”](#) section on page 14-26, you can begin using the DCR APIs to interact with it.

To do this, your application must instantiate an object of class `DCRProxy`. `DCRProxy` is the main DCR abstraction. It provides the methods you need to access DCR device data, and hides communication and request details.

The following topics provide guidelines and examples for using `DCRProxy` and other DCR API classes and methods in application development:

- [About the DCR APIs](#)
- [Creating the DCRProxy Object](#)
- [Creating the APIExtraInfo Object](#)
- [Adding Devices to DCR](#)

CISCO CONFIDENTIAL

- [Updating a DCR Device](#)
- [Adding and Updating Devices in Bulk](#)
- [Retrieving DCR Device Objects](#)
- [Retrieving DCR Devices in Bulk](#)
- [Retrieving Data From a Device Object](#)
- [Comparing Two Device Objects](#)
- [Registering Third-Party Applications with DCR](#)
- [Guidelines for DCR Application Development](#)
- [DCR Error Codes and Interpretations](#)

About the DCR APIs

In addition to DCRProxy, DCR provides the classes shown in [Table 14-14](#).

Note that:

- All DCR APIs write APIs (e.g., addDevice, updateDevice) use CSTM to access DCR. The DCR read APIs (e.g., getDevice, getDevices) communicate directly with CWCS database processes.
- You cannot delete devices from DCR using API calls. Only the administrator can delete devices, and using the DCA interface from the CiscoWorks Home Page.
- All DCR access involving credentials information must be user-authenticated and task-authorized. To assist with this security processing, pass APIExtraInfo with your DCRProxy method calls.
- All credentials data is encrypted while stored and during transport, and DCRProxy decrypts and parses them before passing them to the calling process.
- The DCR APIs are SOAP-enabled, so any non-Java application can make a request using SOAP.

All DCR classes are installed in `$NMSROOT/CSCOpX/lib/classpath/com/Cisco/nm/dcr`. All of them share the `com.cisco.nm.dcr` Java package name.

For DCR API Javadocs, see the [CWCS 3.0 SDK portal](#).

Table 14-14 DCR API Classes

Class	Description
APIExtraInfo	Used with the classes AppID, SourceContext, and SecurityContext to store information about the application that initiated a DCRProxy API call and the form of authentication and authorization each API call will request.
AppId	Stores the name, version and the host name of the machine running the application that initiated the DCRProxy API call.
Attribute	Abstracts a single device attribute and its value, with a complete set of Get and Set methods.
DCRException	Encapsulates standard error/exception handling for all DCR operations.
DCRReturnValues	Represents the return values for bulk Add and Update operations.
DCRUpdates	Encapsulates new, updated and deleted device information.
Device	Abstracts a single device and all of its attributes. This class has a complete set of Get and Set methods.

CISCO CONFIDENTIAL**Table 14-14 DCR API Classes (continued)**

Class	Description
DeviceId	Represents the internal DCR Device ID. Applications use the DCR Device ID in string format.
SecurityContext	Stores the user name, password, Secret Key and Secret User information needed to authenticate or authorize a DCRProxy API call.
SourceContext	Stores information about the application making the DCRProxy API call.

Creating the DCRProxy Object

DCRProxy is the main DCR class. An object of class DCRProxy:

- Represents a DCR Server instance to the application.
- Provides all the methods needed to access and modify data in the DCR Server instance.
- Conceals whether the call is local or remote.
- Provides the CSTM client used to handle communications between DCR and applications.
- Permits registration of HTTP clients.

To start using DCR, your application should instantiate a single DCRProxy object, as shown below:

```
DCRProxy dcrProxy = new DCRProxy() ;
```

Note that creating only a single instance of DCRProxy in the application or session eliminates the overhead required to establish socket-level communication with a local DCR Server.

Also note that DCRProxy uses CSTM classes for logging via log4j. Your application may do so as well. In order for CSTM to log its messages using log4J, either your application or DCRProxy must load the log4J message categories for CSTM (for more on interactions between CSTM and log4j, see the [“Controlling CSTM Logging”](#) section on page 31-3).

The log4j framework permits message categories to be loaded only once per JVM, and in one class loader. This requirement applies irrespective of the number of consumers for CSTM classes. For example: Since both the DCRProxy class and the application-related classes that use CSTM will be loaded in one JVM, the log4J categories for CSTM classes must be loaded either by DCRProxy or by your application class – not by both. If both load the log4J categories for CSTM, then the categories loaded last are the only active categories. If this occurs, log4J will throw exceptions whenever an inactive category object is used for logging, and all logged messages may go to different destinations.

To help you avoid these situations, DCRProxy provides an integer argument that controls the loading of log4J categories for DCR and CSTM classes. You specify this argument at instantiation; for example:

```
DCRProxy dcrProxy = new DCRProxy(0) ;
```

[Table 14-15](#) shows the possible values for the argument and when to use them.

Table 14-15 Controlling log4j Category Loading

Use	When You Want DCR Proxy to Load
DCRProxy(0) ;	The categories for both CSTM and DCR classes. Use this option if your application does not use or load the log4J categories for CSTM classes.

CISCO CONFIDENTIAL**Table 14-15 Controlling log4j Category Loading**

Use	When You Want DCR Proxy to Load
DCRProxy(1) ;	The categories for DCR classes only. It will not load the categories for CSTM classes. Use this option if your application uses CSTM and loads the log4J categories for CSTM classes.
DCRProxy(2) ;	No categories at all. This option assumes that your application loads the categories for both the DCR and CSTM classes.

Note that `DCRProxy(2)` will not work for any web-based application context that runs under Tomcat. In this case, the application will not be able to load the categories for DCR classes because the DCR classes are loaded via the class loader, which is different from an application context-specific class loader. If your application runs under Tomcat, use either `DCRProxy(0)` or `DCRProxy(1)`, depending on whether or not you are having the application load the log4J categories for CSTM classes.

If you pass any value other than those shown in [Table 14-15](#) to this constructor, `DCRProxy` will reset the argument value to 0 and load the categories for both CSTM and DCR classes. The default constructor, `DCRProxy()`, assumes the argument value is 0 and loads the categories for both CSTM and DCR classes. Use the default constructor if your application does not use or load the log4J categories for CSTM classes.

Creating the APIExtraInfo Object

Most `DCRProxy` API calls accept objects of class `APIExtraInfo` in addition to their regular arguments. `APIExtraInfo` encapsulates:

- The `AppID`. This object is the unique ID of your application, and establishes your application name, version number, and host information for use in the `SourceContext` object.
- The `SourceContext`. This object establishes your application and its context as the source of the `DCRProxy` API call.
- The `SecurityContext`. This object contains the information needed to authenticate or authorize the `DCRProxy` API request. Applications residing on the same machine as the DCR Server have access to the local version of the DCR APIs, and need pass only the requesting username. Applications on remote servers must use the remote (or “north-bound”) versions of the DCR APIs, and must pass the requesting username and password.

[Example 14-2](#) shows a modifiable code fragment that specifies this information. You will need to change this code before using it (i.e., comment out one of the two `SecurityContext` alternatives, and choose to pass either the password or secret key value, but not both, for remote API calls).

Example 14-2 Instantiating APIExtraInfo

```
AppId Myapp = new AppId("Device Manager", //unique AppID and application name
    "1.3.2", //application version number
    "192.168.1.15"); // host on which the application is running
SourceContext source = new SourceContext(Myapp) ;
// For Local API calls
SecurityContext security = new SecurityContext("username");

// For North-Bound API calls
SecurityContext security = new SecurityContext("username",
    "password",
```


CISCO CONFIDENTIAL

```

        "secretKey")
//Wrapper for Source and Security information:
APIExtraInfo extraInfo = new APIExtraInfo(security,
        source);

```

Adding Devices to DCR

Example 14-3 shows sample code for adding a standard (non-proxy) device to DCR using the `addDevice()` call.

Example 14-3 Adding a Standard Device

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.STANDARD_DEVICE);
device.SetAttribute("display_name", "Bldg X Floor Y Switch 1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "1234567");
// set other attributes
DeviceId id = null;
try
{
    id = dcrProxy.addDevice(device,extraInfo);
}
catch (DCRException de)
{
    System.out.println("Error in adding Device " +
        de.getMessage());
    // Handle any error from DCR
}
catch (Exception e)
{
    System.out.println("Error in adding Device " +
        e.getMessage());
    // Do application-specific things
}
System.out.println("ID for New Device = "+ id.getValue());
// This is the value that the application will retain

```

To add other types of devices using the same call, simply change the `SetDCRDeviceCategory` and `SetAttribute` portions of the code as appropriate for each device, as shown in the following examples.

Example 14-4 Adding an AUS Device

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.AUS_DEVICE);
device.SetAttribute("display_name", "AUS 1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "UNKNOWN");
// set other attributes
DeviceId id = null;
...

```

CISCO CONFIDENTIAL**Example 14-5 Adding an AUS-Managed Device**

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.STANDARD_DEVICE);
device.SetAttribute("parent_aus_id", "999999");// Set the parent AUS id
device.SetAttribute("display_name", "AUS Managed Device 1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "1234567890");
// set other attributes
DeviceId id = null;
...

```

Example 14-6 Adding a DSBU Cluster

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.DSBU_DEVICE);
device.SetAttribute("display_name", "DSBU-Cluster 1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "UNKNOWN");
device.SetAttribute("mdf_type", "278283831");
// set other attributes
DeviceId id = null;
...

```

Example 14-7 Adding a DSBU Cluster Member

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.STANDARD_DEVICE);
device.SetAttribute("parent_dsbu_id", "110");// Where "110" is the parent DSBU Cluster ID
device.SetAttribute("display_name", "DSBU-Cluster Managed Device 1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "122333");
// set other attributes
DeviceId id = null;
...

```

Example 14-8 Adding a CNS Configuration Engine (CNS Server)

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.CNS_DEVICE);
device.SetAttribute("display_name", "CNS_Server_1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "UNKNOWN");
device.SetAttribute("mdf_type", "277587376");

// set other attributes
DeviceId id = null;
...

```

CISCO CONFIDENTIAL**Example 14-9 Adding a CNS managed device**

```

Device device = new Device();
device.SetDCRDeviceCategory(Device.STANDARD_DEVICE);
device.SetAttribute("parent_cns_id", "110");// Where "110" is the parent CNS Server ID
device.SetAttribute("display_name", "CNS_managed_1");
device.SetAttribute("management_ip_address", "1.2.3.2");
device.SetAttribute("sysObjectID", "UNKNOWN");

// set other attributes
DeviceId id = null;
...

```

Updating a DCR Device

You can use code like that shown in [Example 14-10](#) to update attributes or credentials for any existing DCR Device.

Example 14-10 Updating a DCR Device

```

DeviceId deviceID = new DeviceId("<known-device-id>");
Device device = new Device(deviceID);
// Set new attribute values
device.SetAttribute("display_name", "Device 2 New");
// set other attributes

try
{
    dcrProxy.updateDevice(device,extraInfo);
}
catch (DCRException de)
{
    System.out.println("Error in updating Device " +
        de.getMessage());
    // Do application specific things
}
catch (Exception e)
{
    System.out.println("Error in updating Device " +
        e.getMessage());
    // Do application specific things
}

```

Adding and Updating Devices in Bulk

These two APIs let you add or update more than one device object at a time:

- `public DCRReturnValues addDevices(Device[] devices, APIExtraInfo apiExtraInfo) throws DCRException`
- `public DCRReturnValues updateDevices(Device[] devices, APIExtraInfo apiExtraInfo) throws DCRException`

CISCO CONFIDENTIAL

Both APIs return an object of class `DCRReturnValue`, which holds all information about the operation on each device. The information is stored in two arrays: one for the `DeviceId` objects, and the second for operation error codes. The length of each array is always same as the number of `Device` objects you pass in the API call. The objects in these arrays correspond to each `Device` object in the `devices` array that you pass.

To get this information from the appropriate arrays in `DCRReturnValue`, use the following methods:

- `public DeviceId getDeviceId(int index)` returns the `DeviceId` object at the index you pass. Use this method after the `addDevices` call to retrieve and process all the newly created Device IDs.
- `public int getErrorCode(int index)` returns the error code associated with the operation on the `Device` object at the index you pass.

[Example 14-11](#) shows typical code for a bulk addition. A bulk update would use the different method but have essentially the same structure.

Example 14-11 Adding Devices in Bulk

```

        DCRReturnValues drv = null;

        // Call add API and collect its output in above DCRReturnValues object
...

int nErrorCode;
if(drv != null)
{
    for(int nLoop = 0; nLoop < numberOfDevicesAdded; nLoop++)
    {
        // check if device was added successfully
        // If return array contains valid DeviceId, this means that device was added successfully
        if(drv.getDeviceId(nLoop) != null) continue; // Valid DeviceId returned

        // Otherwise retrieve error code
        nErrorCode = drv.getErrorCode(nLoop);

        // The error code represents the ID of DCRException.

        // Handle the error here ...

        }// end for
    }// end if

```

Retrieving DCR Device Objects

[Example 14-12](#) demonstrates how to retrieve selected DCR Device Objects and their data using a list of DCR Device IDs supplied by an update event.

Example 14-12 Retrieving Device Objects

```

DeviceId[] deviceIDs = <populate device ids...>

String[] requiredAttributes = { "display_name",
                                "management_ip_address",

```

CISCO CONFIDENTIAL

```

        "snmp_v2_ro_comm_string"
        };

Device[] devices = null;

try
{
    devices = dcrProxy.getDevices(deviceIDs,
        requiredAttributes,extraInfo);
}
catch (DCRException de)
{
    System.out.println("Error in getting Devices " +
        e.getMessage());
    // Do application-specific things
}
catch (Exception e)
{
    System.out.println("Error in getting Device " +
        e.getMessage());
    // Do application specific things
}

```

Retrieving DCR Devices in Bulk

[Example 14-13](#) demonstrates how to fetch a large number of DCR Devices at once. This code follows the guidelines given in the “[Enhancing DCR Performance](#)” section on page 14-49, specifically those for handling groups of devices with more than 5,000 members. Note that the DeviceIdIterator is a class in the package com.cisco.nm.dcr.

Example 14-13 Retrieving Devices in Bulk

```

DeviceId[] deviceIDs = <the array of DeviceId to be fetched>
...
Device[] devices = null;
Vector v = null;

try
{
    DeviceIdIterator it = new DeviceIdIterator(deviceIDs);
    DeviceId[] onlyFewDeviceIDs = null;

    while(it.hasNext())
    {
        onlyFewDeviceIDs = it.next();
        devices = _dcrProxy.getDevices(onlyFewDeviceIDs, attributes, aei);
        if(devices != null)
        {
            if(v == null) v = new Vector();
            v.addAll(Arrays.asList((Object[])devices));
        }
    } // end while
}
catch(DCRException ex)
{
    // Handle exception
}
catch(Exception ex)

```

CISCO CONFIDENTIAL

```

    {
        // Handle exception
    }

    if(v == null) return null;
    v.trimToSize();
    if(v.size() <= 0) return null;

    devices = new Device[v.size()];
    v.toArray(devices);

    // Here the devices object contain all the devices

```

Retrieving Data From a Device Object

[Example 14-14](#) shows a couple of ways to get specific device data from a device object. This example assumes you have already retrieved one or more device objects using code such as the example shown in the [“Retrieving DCR Device Objects”](#) section on page 14-34.

Example 14-14 Retrieving Specific Device Data

```

Device device = <fetched device via DCR API ...>
//Get a single Attribute or Credential value
Attribute attr = device.GetAttribute("snmp_v2_ro_comm_string");
// Note that attribute object may be null
if(attr != null)
{
    String strValue = attr.getValue();
    ...
}
// Alternate code: Get a single Attribute or Credential value
if(device.isAttributeAvailable("snmp_v2_ro_comm_string"))
{
    Attribute attr = device.GetAttribute("snmp_v2_ro_comm_string");
    String strValue = attr.getValue();
    ...
}
//Alternate code: Get all attributes for each device
Iterator it = device.GetAttributes();
//Then use the it iterator to cycle through all the attributes for the current device

```

Two methods you can use for retrieving specific device data will return special codes:

- `public int GetDCRDeviceCategory()` returns the device category of a device added with any valid value for `dcr_device_type`. The possible return values are:
 - 0 for a standard device
 - 1 for a Logical DSBU Cluster
 - 3 for an Auto Update Server (AUS)
 - 4 for a CNS Configuration Engine (CNS)
- `public int GetDeviceAccessType()` returns an integer indicating how the device is or should be accessed. The possible return values are:

CISCO CONFIDENTIAL

- 0 means no access type, or the device is a standard device, or the access information is not available.
- 1 means the device is part of a DSBU cluster and can be accessed as a member of a DSBU cluster. Such device objects will also have a 'parent_dsbu_id' attribute that is the Device ID of a logical DSBU cluster in DCR.
- 3 means the device is accessed via AUS. Such device objects will also have a parent_aus_id attribute that is device ID of a Auto Update Server in DCR.
- 4 means the device is accessed via CNS. Such device objects will also have a parent_cns_id attribute that is device ID of a CNS Server in DCR.

Note that this method assumes that complete device information (specifically, the parent DSBU and AUS ID attributes) have been fetched from DCR. If not, this method will always return 0.

Comparing Two Device Objects

The following code fragment shows how to compare two device objects. Note that two DCR devices are equal if their DCR Device ID values are the same and their attributes and attribute values are the same.

```
DeviceId id = device.GetID();
device.equals(device2Object);
```

Registering Third-Party Applications with DCR

Although CWCS-based applications use ESS to get events indicating updates to DCR devices, third-party applications must listen for events using HTTP. [Example 14-15](#) shows how to register a third-party application so DCR will send events to it.

Example 14-15 Registering to Receive DCR Events Via HTTP

```
try
{
    dcrProxy.registerForHTTPEvents (appID,
        "<application-URL-to-receive-events>"
        extraInfo);
}
catch (DCRException de)
{
    System.out.println("Error in registering app " +
        e.getMessage());
    // Do application specific things
}
catch (Exception e)
{
    System.out.println("Error in registering app " +
        e.getMessage());
    // Do application specific things
}
}
```

CISCO CONFIDENTIAL**Guidelines for DCR Application Development**

Keep the following guidelines in mind when creating applications that use data from DCR:

- DCR never communicates with devices directly.
- DCRProxy conceals all communications details. You do not need to know the DCR Server mode or any other communications details to use DCRProxy successfully.
- Your application must be listening for DCR events in order to receive them.
- Your application is responsible for filtering events for devices that are not applicable to it. DCR itself performs no filtering.
- When filtering, do not rely solely on the sysObjectID or mdf_type value contained in the event to identify the device for you. Since DCR relies on many applications to populate its device list, these attributes cannot be guaranteed to have values other than “unknown”. Your application should test for situations where these attributes have the value “unknown”, and attempt to identify the device either by calling the device directly or by other means.
- Your application is not allowed to delete the devices in DCR. In Master/Slave setups, these device deletions are propagated to all DCR Slave applications, which can seriously affect the operation of other applications sharing the DCR data.
- Your application may cache device credentials in memory, but for security reasons, should never persist or save credentials to disk.
- Your application should always check the availability of DCR for updates before performing critical operations, such as changing device credentials.
- DCR APIs do not validate the values of attributes and credentials while adding and updating devices.
- You cannot leave both sysObjectID and mdf_type attributes blank, but you can specify either one or both of them as “unknown”. DCR will determine the sysObjectID value automatically if only the mdf_type is specified, and vice versa. If you enter both, DCR will *not* check that they map properly.
- If you maintain an application-specific store of device information, including device references and other data, remember:
 - You will need to maintain a mapping between your application-specific device identifiers and the DCR Device IDs. This is required because your application can only fetch device details from DCR by using the DCR Device ID.
 - Your application must maintain a “last updated” timestamp value for each device in its application-specific device store. This is needed so that your application can get device updates from DCR using the getNewDevices() API call.
 - Your application should always filter incoming DCR events before populating the application-specific device list with new entries or presenting it to users.

DCR Error Codes and Interpretations

The DCR error codes and what they stand for are as follows:

Table 14-16 DCR Error Codes and Interpretations

Error Number	Error String	Meaning
0	NO_ERROR	No error.
-1	UNKNOWN_EXCEPTION	Unknown error occurred.

CISCO CONFIDENTIAL**Table 14-16 DCR Error Codes and Interpretations (continued)**

Error Number	Error String	Meaning
-3	NO_ID_ATTRIBUTES_SPECIFIED	At least one mandatory attribute was not specified.
-4	MORE_PARENT_IDS_SPECIFIED	More than one Parent ID was specified.
-5	DEVICE_RECORD_EXISTS	Duplicate device. Record for specified device already exists.
-6	DUPLICATE_DEVICE_NAME	Specified device name already exists.
-7	UNSUPPORTED_ARGUMENT_VALUE	Argument value is not supported.
-8	NO_SUCH_DEVICE	Specified device does not exist in records.
-9	DEVICE_EXCLUDED	Device is marked as excluded.
-10	DEVICE_ID_FORMAT_ERROR	Invalid device ID format.
-11	DEVICE_ID_COUNT_ERROR	Internal error during device ID count.
-12	DUPLICATE_DEVICE_NAME_UNDER_PARENT	Duplicate device ID under the same Parent.
-13	NO_DEVICE_NAME	No device name specified.
-15	DUPLICATE_ATTRIBUTE	User defined field with the same name already exists.
-16	INVALID_APP_ID	Invalid application ID.
-17	APP_ID_COUNT_ERROR	Internal error during application ID count.
-18	DUPLICATE_DEVICE_RECORD	Duplicate record found for this device.
-19	AUTHENTICATION_FAILED	User is not authenticated.
-20	DEVICE_AUTHORIZATION_FAILED	User is not authorized to perform this task on the device.
-21	EXCLUDE_FILE_LOAD_ERROR	Error in loading exclude file. Check the file name and path.
-22	SYSOID_VALUE_ERROR	Must specify sysObjectID/MDF type.
-23	CTM_COMM_ERROR	Error in communicating with DCA Server.
-24	NO_DSBU_MEMBER_NUMBER_SPECIFIED	DSBU Cluster member number not specified.
-25	INVALID_ATTRIBUTE_INFO	Invalid attribute specified.
-26	DUPLICATE_MEMBER_UNDER_PARENT	DSBU Cluster member number already exists.
-27	DCR_MODE_IS_NOT_SLAVE	The DCR mode is not set as Slave.
-28	DCR_MODE_IS_NOT_MASTER	The DCR mode is not set as Master.
-29	MASTER_DATA_RESTORED	Master data is restored.
-30	INVALID_SLAVE	The specified slave is not valid.
-31	NUM_ATTRIBUTE_LIMIT	Attribute limit .
-33	TASK_AUTHORIZATION_FAILED	User is not authorized to perform this task .
-34	CAM_EXCEPTION	Internal error occurred during security checks.
-35	AUS_DEVICE_ID_NOT_SPECIFIED	AUS device ID is not specified.
-36	INVALID_ATTR_VALUE	One of the attribute values is invalid.
-37	ERROR_IN_OPENING_EXCLUDE_FILE	Error in opening exclude file. Check file name and path

CISCO CONFIDENTIAL**Table 14-16 DCR Error Codes and Interpretations (continued)**

Error Number	Error String	Meaning
-38	ERROR_IN_PARSING_EXCLUDE_FILE	Error in parsing exclude file. Please check the format of the file.
-39	CANNOT_CHANGE_MASTER	Error in configuring the slave because domain IDs of the slave and master are the same. Change mode to Standalone first and then change it to Slave.
-40	CANNOT_CONFIG_MASTER	Error in configuring the slave because domain IDs of the slave and master are the same .
-41	INVALID_DCR_DEVICE_TYPE	The specified device type is not valid.
-42	INVALID_ATTR_VAL_LENGTH	The attribute value length is not valid.
-43	INVALID_CRED_VAL_LENGTH	The credential value length is not valid.
Database Related		
-101	DATABASE_EXCEPTION	Error in database.
-102	PRIMARY_KEY_NOT_UNIQUE	Primary key specified already exists.
-103	SQL_SYNTAX_ERROR	Syntax error in the SQL statement.
CSTM related		
-201	DCR_SERVER_NOT_RUNNING_ERROR	DCA Server is not running.
-202	CSTM_INTERNAL_ERROR	Internal error in communication channel.
-203	ERROR_IN_COMM_WITH_SERVER	Error in communicating with Peer Server. Ensure that self-signed certificate is generated or copied correctly and System Identity Setup is done correctly.
-211	PEER_DCR_SERVER_NOT_RUNNING_ERROR	Peer DCA Server is not running.
-213	ERROR_IN_COMM_WITH_PEER_SERVER	Error in communicating with Peer Server. Ensure that self-signed certificate is generated or copied correctly and System Identity Setup is done correctly.
-221	PEER_DCR_SECURITY_SERVER_NOT_RUNNING_ERROR	Peer DCR security server is not running.
-223	ERROR_IN_COMM_WITH_PEER_DCR_SECURITY_SERVER	Error while communicating with Peer DCR Security Server.

Responding to DCR Events

Example 14-16 shows code for a sample application that can respond to DCR Device and DCR RESTORE Events.

**Note**

This code is incomplete. It is provided as an *illustration* of application-side DCR event processing only.

Example 14-16 Responding to DCR Device and RESTORE Events

CISCO CONFIDENTIAL

```

public class DCREventReceiver {

    // Instantiate Tibco receiver and subscribe to "cisco.mgmt.cw.cmf.dcr" event
    public DCREventReceiver() {

        try {
            _parser = SAXParserFactory.newInstance().newSAXParser();
            _handler = new SXP();
            log.info("INFO: Creating SAX Parser object");
        } catch (Exception ex) {
            log.fatal("Error creating: " + ex.getMessage());
        }

        try {
            TopicConnectionFactory factory;
            factory = new TopicConnectionFactoryImp();
            _con = factory.createTopicConnection();
            _session = _con.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        } catch (JMSEException e) {
            log.fatal(
                "ERROR: cannot connect to Topic for receiving events.");
            log.fatal(MoMUtils.getStackTrace(e));
            return;
        }

        log.info("Listening on the topic: " + "cisco.mgmt.cw.cmf.dcr");
        try {
            Topic topic = _session.createTopic("cisco.mgmt.cw.cmf.dcr");
            _sub = _session.createSubscriber(topic);
            _sub.setMessageListener(this);
        } catch (JMSEException e) {
            log.fatal("ERROR: Cannot create the Topic listener.");
        }
    }

    public void processEvents() {
        try {
            _con.start();
            log.info("INFO: Starting to receive message");
        } catch (JMSEException e) {
            log.fatal("ERROR: cannot process events from topic");
            log.fatal(MoMUtils.getStackTrace(e));
        }
    }

    public void onMessage(Message jmsMsg) {
        log.info("Calling on message to create a thread.... ");
        try {
            TextMessage text = null;
            if (jmsMsg instanceof TextMessage) {
                text = (TextMessage) jmsMsg;
            } else {
                log.debug("Message is not TextMessage: " + jmsMsg);
                return ;
            }

            String xmlData = text.getText();
            log.info("Inserting event in queue: " + xmlData);
            processMessage(xmlData);
        } catch (Exception exception) {
            log.fatal("Problem in onMessage " + exception);
        }
    }
}

```

CISCO CONFIDENTIAL

```

// process the Event data
//
void processMessage(String txt) {
    dcrDeviceVec.clear();
    dcrEventType = "";
    byte[] txtBytes = txt.getBytes();
    ByteArrayInputStream bis = new ByteArrayInputStream(txtBytes);
    try {
        _parser.parse(bis, _handler);
    } catch (Exception ex) {
        log.fatal("Error parsing event data: " + txt
            + ". Reason: " + ex.getMessage());
        return;
    }

    for (int i = 0; i < dcrDeviceVec.size(); i++) {
        // Device data
    }

    log.info("Event type is"+ dcrEventType);
}

class SXP extends DefaultHandler {
    public void startDocument() throws SAXException {
        log.info("SXP: StartDocument method is called");
    }

    public void endDocument() throws SAXException {
        log.info("SXP: EndDocument method is called");
    }

    public void startElement(String namespaceURI, String sName,
        String qName, Attributes attrs) {
        log.info("SXP: StartElement method is called " );
        log.info("SXP: NameSpaceURI is = " + namespaceURI);
        log.info("SXP: simple Name is = " + sName);
        log.info("SXP: Qualified Name is = " + qName);

        for (int i = 0; i < attrs.getLength(); ++i) {
            log.info("SXP: Local Name is = "
                + attrs.getLocalName(i));
            log.info("SXP: QName is = "
                + attrs.getQName(i));
            log.info("SXP: Attribute value is = "
                + attrs.getValue(i));
        }

        if (qName.equalsIgnoreCase("Device")) {
            processDevice = true;
        }
    }

    public void endElement(String namespaceURI, String sName, String qName) {
        log.info("SXP: EndElement method is called " );
        log.info("SXP: NameSpaceURI is = " + namespaceURI);
        log.info("SXP: simple Name is = " + sName);
        log.info("SXP: Qualified Name is = " + qName);

        // for now, ignore bulk events, restore events etc.

        if (dcrEventType.equals("BULK_DEVICES_ADDED") ||
            dcrEventType.equals("BULK_DEVICES_DELETED") ||
            dcrEventType.equals("BULK_DEVICES_UPDATED") ||
            dcrEventType.equals("DCR_DATA_RESTORED_FROM_DIFFERENT_DOMAIN") ||

```

CISCO CONFIDENTIAL

```

        dcrEventType.equals("DCR_DATA_RESTORED")) {
            processDevice = false;
            return;
        }

        if (!processDevice)
            return; // ignore devices we do not want

        for (int i = 0; i < dcrEventFieldNames.length; i++) {
            dcrDeviceFields[i] = "";
            if (qName.equalsIgnoreCase(dcrEventFieldNames[i])) {
                if (value != null && !value.equals(""))
                    dcrDeviceFields[i] = value.toString();
            }
            if (qName.equalsIgnoreCase("SysObjectId")) {
// Is the device is supported by the application
//
                if (! applicationSupportedType(dcrDeviceFields[i])) {
                    processDevice = false;
                    return;
                }
            }
            // skip if the device id is not in our database
            // for a delete event.
            if (qName.equalsIgnoreCase("DeviceId")) {
                if (dcrEventType.equals(DCR_DEVICES_DELETED)) {
                    synchronized(application.deviceIdsHash) {
                        if
(!application.deviceIdsHash.containsKey(dcrDeviceFields[i])) {
                            processDevice = false;
                            return;
                        }
                    }
                }
            }
        }

        if (qName.equalsIgnoreCase("Device") && processDevice) {
            // store all device attributes in an array
            dcrEventType = dcrDeviceFields[0];
            String dcrDeviceId = dcrDeviceFields[1];
            String sysObjectId = "";
            String ipAddress = "";
            String hostName = "";
            String displayName = "";
            long dcrTransactionId = 0;
            if (!dcrEventType.equals("DEVICES_DELETED")) {
                // delete event has only dcrId.
                sysObjectId = dcrDeviceFields[2];
                ipAddress = dcrDeviceFields[3];
                hostName = dcrDeviceFields[4];
                displayName = dcrDeviceFields[5];
                dcrTransactionId = Long.parseLong(dcrDeviceFields[6]);
            }

// Application-specific processing here...
        }

        public void characters(char[] ch, int start, int length) {

            log.info("SXP: Characters method is called");
            String s = (new String(ch,start,length) ).trim();
            log.info("SXP: The value is " + s);
        }
    }

```

CISCO CONFIDENTIAL

```

        if (value == null) {
            value = new StringBuffer(s);
        } else {
            value.append(s);
        }
    }

} // End of Inner class - SXP

//Vector to store the list of device Info
Vector devDataVec=null;
StringBuffer value=null;
String dcrEventType = null;

// parsing specific variables
DefaultHandler _handler = null;
SAXParser _parser = null;

public static final Class _Class = DCREventReceiver.class;

private TopicConnection _con = null;
private TopicSession _session = null;
private TopicSubscriber _sub = null;
private SharedQueue _queue = null;
Vector dcrDeviceVec = null;
boolean processDevice = false;
String[] dcrDeviceFields = null;
Logger log;
}**

```

Using DCR Domain and Transaction IDs

Example 14-17 shows code for a sample application that checks the Domain and Transaction at startup.

**Note**

This code is incomplete. It is provided as an *illustration* of application-side DCR event processing only.

Example 14-17 Using the DCR Domain and Transaction IDs During Startup

```

//initiate the process of listening for events from the DCR
dcrEventListener = new DCREventReceiver();
dcrEventListener.processEvents();

// get the latest transaction ID from the application database
public long getMaxDcrTransactionId() {

}

// Filter device IDs; get only application-specific device IDs from DCR
public Vector getFilteredDcrDevices(String action, DeviceId[] dcrDevices) {
for (int i = 0; i < dcrDevices.length; i++) {
    DeviceId meDev = dcrDevices[i];
    if (action.equals("DEVICES_DELETED")) {
        // only device ID will be in the event
        String dcrDeviceId = meDev.getValue();
    } else { // extract more event details from payload
        find out sysObjectId or mdf_type of the device...
    }
}
}

```

CISCO CONFIDENTIAL

```

        find out whether application supports this specific device
    }
}
return devicesVec;
}

/** this routine reads DCR data upon startup and syncs up data */
/** in preparation for startup synching with DCR to catch any offline updates */

getDCRUpdatesOnStartup(long MaxDcrTransactionId){

    Device dcrDev = new com.cisco.nm.dcr.Device();
    DeviceId[] ids = null;
    Vector allIdsVec = new Vector();
    Vector newDevicesVec = null,
          updatedDevicesVec = null,
          deletedDevicesVec = null;

    String dcrDomainId = dcr.getDCRDomainID(extraInfo);

    String oldDcrDomainId = get dcrDomainId from application datatbase

    if (oldDCRDomainId not availble in application database) // first time
        store dcrDomainId in application database
    } else {
        if (!oldDCRDomainId.equals(dcrDomainId)) {
            clean application database and start resync DCR data or
            indicate application about the change
            application specific action
        }
    }

    if (appMaxDcrTransactionId == 0) { //first time startup -
        get devices from DCR through
        Devices[] devices = getNewDevices(appMaxDcrTransactionId,extraInfo)
        filteredDevices = getFilteredDcrDevices("DEVICES_ADDED",
updateInfo.getNewDevices());
        update application managed list
        return ;
    }

    // If not, check if DCR transaction ID is different
    // If DCR transaction ID is different, adds/deletes/updates took
    // place when application was offline. Get these changes now.
    // Load existing information from application database.
    //
    long maxDcrTransactionId = dcr.getMaxTransactionID(extraInfo);
    if (maxDcrTransactionId == appMaxDcrTransactionId)
        return null;
    } else (if appMaxDcrTransactionId < maxDcrTransactionId ) {

        // Otherwise, there were updates when application was offline.
        // Retrieve the current device IDs from application and then start sync.

        DCRUpdateData updateInfo = dcr.getDCRUpdates(appMaxDcrTransactionId,
                                                    dcrDomainId, dcrDeviceIds, extraInfo);
        newDevicesVec = getFilteredDcrDevices("DEVICES_ADDED",
updateInfo.getNewDevices());
        deletedDevicesVec = getFilteredDcrDevices("DEVICES_DELETED",
updateInfo.getDeletedDevices());
        updatedDevicesVec = getFilteredDcrDevices("DEVICES_UPDATED",
updateInfo.getUpdatedDevices());
        // Updated application managed list based on the new, deleted and updated data from DCR.

```

CISCO CONFIDENTIAL

```

}
}
/*****
main method for the application.
*****/
public static void main(String[] args) throws Exception {
// initialize first: Start new thread to listen to DCR events, and push to the queue.
startDCREventListener();
    long MaxDcrTransactionId = application.getMaxDcrTransactionId();
// Now start threads to handle all device processing.
    syncWithDCRAtStartup(MaxDcrTransactionId);
}
}

```

Using the DCR Command-Line Interface

The DCR command-line interpreter allows you to conduct via the command line most of the important tasks normally accessible only via the DCR API or the GUI. As with the GUI, all CLI commands must be executed on the same machine on which the target DCR server is running.

Table 14-17 shows the commands and their usage.

To start the DCR command line interpreter:

1. Change to the `NMSROOT/bin` folder
2. Execute `dcrccli`.
3. At the prompt, enter a valid Cisco Works user ID and password. When `dcrccli` finishes authenticating this user name, it displays the `dcrccli>` prompt.

Table 14-17 DCR CLI Commands

Command	Syntax & Description
add	<p>add ip=value hn=value di=value dn=value -a attrname=value</p> <p>Adds the specified device to the DCR server's device list. You must specify an IP address (ip), host name (hn) or device Identity(di). You must also specify the the Display Name (dn) and the Attribute name (-a attrname). The attribute sysObjectID is mandatory. You can specify as many comma-separated attribute name=value pairs as needed.</p> <p>For example:</p> <p>add ip=1.1.1.1 hn=device1 dn=cisco.com -a sysObjectID=1.3.6.1.4.1.9.1.6</p>
mod	<p>mod id=value ip=value hn=value di=value dn=value -a attrname=value</p> <p>Modifies the specified device. You must enter the following:</p> <ol style="list-style-type: none"> 1. Enter the Device ID (id). 2. Enter either the IP Address (ip), Hostname (hn), or Device Identity (di). 3. Enter the Display Name (dn) and the Attribute name (-a attrname). You can add multiple attributes. <p>For example:</p> <p>mod id=54341 ip=2.2.2.2 dn=cisco.com -a display_name=new_name</p>

CISCO CONFIDENTIAL**Table 14-17 DCR CLI Commands (continued)**

Command	Syntax & Description
del	<p><code>del id=value</code></p> <p>Deletes the specified device. You must specify the Device ID (id).</p> <p>For example:</p> <pre>del id=256666989</pre>
impAcs	<p>mpACS ot=OS Type hn=ACS Server Name or IP address un=ACS admin user name pwd=ACS admin password prt=port number cr=conflict resolution option</p> <p>Import sdevice information directly from a remote ACS system.</p> <p>You must specify Operating System Type [ot], ACS Server Name or IP address [hn], ACS admin user name [un], ACS admin password [pwd] and port number [prt].</p> <p>The default port number is 2002.</p> <p>Conflict resolution option in dcrecli that helps to override dcr data from import source file,rnms,nms,acs). dcrecli will have an extra cr (conflict resolution) option added to the end of all the import commands. This option will take two values viz. dcr for keeping the dcr data and {file, nms, rnms, acs} for keeping the data from import source.</p> <p>If we are not specifying cr option explicitly, dcr will be taken as the default value for cr option for all the import commands.</p> <p>Example:</p> <p>impAcs ot=WIN2K hn=1.2.3.4 un=acsadmin pwd=acspwd prt=2002</p>
impFile	<p>impFile fn=file name ft=file type cr=conflict resolution option</p> <p>Imports device information from a file. You must specify a filename(fn) with complete path, and the file type (ft). CSV and XML are the valid values for file type.</p> <p>Conflict resolution option in dcrecli that helps to override dcr data from import source file,rnms,nms,acs). dcrecli will have an extra cr (conflict resolution) option added to the end of all the import commands. This option will take two values viz. dcr for keeping the dcr data and {file, nms, rnms, acs} for keeping the data from import source.</p> <p>If we are not specifying cr option explicitly, dcr will be taken as the default value for cr option for all the import commands.</p> <p>Example:</p> <pre>impFile fn=d:/mypath/myImportFile.xml ft=xml cr={dcr file}</pre>
impNms	<p>impNms nt=NMS type il=Installation location cr=conflict resolution option</p> <p>Imports device information directly from a local NMS.</p> <p>You must specify the NMS type [nt]. Valid values for NMS Type are are HPOV6.x and Netview7.x.</p> <p>Conflict resolution option in dcrecli that helps to override dcr data from import source file, rnms, nms, acs). dcrecli will have an extra cr (conflict resolution) option added to the end of all the import commands. This option will take two values viz. dcr for keeping the dcr data and {file, nms, rnms, acs} for keeping the data from import source.</p> <p>If we are not specifying cr option explicitly, dcr will be taken as the default value for cr option for all the import commands.</p> <p>Example:</p> <p>impNms nt=HPOV6.x il=/opt/OV</p>

CISCO CONFIDENTIAL**Table 14-17 DCR CLI Commands (continued)**

Command	Syntax & Description
impRNms	<p>impRNms nt=NMS type hn=hostname un=Remote User Name il=Installation location ot=OS Type cr=conflict resolution option</p> <p>Imports device information directly from a remote NMS.</p> <p>You must specify the NMS type [nt], Remote Host Name or IP address [hn], Remote User Name [un], Installation location of the NMS [il], and OS Type[ot].</p> <p>Valid values of NMS Type are HPOV6.x and Netview7.x. You can specify your OS types as HPUX, AIX, or SOL.</p> <p>Conflict resolution option in dcrcli that helps to override dcr data from import source file, rnms, nms, acs). dcrcli will have an extra cr (conflict resolution) option added to the end of all the import commands. This option will take two values viz. dcr for keeping the dcr data and {file, nms, rnms, acs} for keeping the data from import source.</p> <p>If we are not specifying cr option explicitly, dcr will be taken as the default value for cr option for all the import commands.</p> <p>Example: impRNms nt=HPOV6.x hn=1.2.3.4 un=root il=/opt/OV ot=SOL</p>
exp	<p>exp fn=filename ft={csv xml}</p> <p>Exports the current DCR device list to a file in CSV or XML format. You must specify a filename with complete path, and whether the file type is in CSV or XML format. For example:</p> <p>exp fn=d:/mypath/myExportFile.xml ft=xml</p>
lsids	<p>lsids {all dn=displayName ip=IPAddress}</p> <p>Lists the DCR device ID for devices stored on the DCR Server. It will list all devices if you specify no additional parameters [KR: or must you specify lsids all in order to list all?], or only the device ID for the device with the specified display name or IP address. If several devices share the same IP address, the command will list the device IDs for all of them. For example:</p> <p>lsids ip=168.192.1.20</p>
detail	<p>detail id=deviceID</p> <p>Lists all the details about the device with the ID you have specified.</p> <p>For example: detail id=89992921023</p>
lsattr	<p>lsattr</p> <p>This lists Attribute Name, Attribute Description, and Attribute Type.</p> <p>Attribute Type is a constant that identifies an Attribute Name. For example, Attribute Type 1072 identifies the attribute name display_name. For example, Attribute Type 1072 identifies the attribute name display_name.</p>
lsmode	<p>lsmode</p> <p>Lists the DCR ID, the DCR Group ID, the current DCR mode, and the associated Master or Slaves.</p>
setmaster	<p>setmaster</p> <p>Sets the DCR server to Master mode. The command takes no parameters.</p>
setstand	<p>setstand</p> <p>Sets the DCR server to Standalone mode. The command takes no parameters.</p>

CISCO CONFIDENTIAL**Table 14-17 DCR CLI Commands (continued)**

Command	Syntax & Description
setslave	<pre>setslave master=DCRGroupID port=portNumber</pre> <p>Sets the DCR server to Slave mode. You must specify the DCR Group ID for the new Master with which this slave will communicate. You must also specify the Master's port number if it is anything except 443 (443 is the default Master port). For example:</p> <pre>setslave master=DCRMaster221 port=1099</pre>
exit	Exits the DCR command line interpreter shell.

Enhancing DCR Performance

A number of DCR APIs can cause performance problems under certain circumstances. To help you avoid these problems, the CWCS team makes the following recommendations:

1. Avoid the `getDevices(ApiExtraInfo extranifo)` method.

This particular method returns the complete details of all attributes and credentials of all devices in DCR. This API should be used only very rarely, if at all, since there are very few cases when an application will want to retrieve that much information with a single call. If your DCR server contains a very large number of devices, use of this API can consume large amounts of memory, and increase the risk of CSTM errors, serialization/deserialization issues, and database-connection time out errors. This is compounded if several applications are using the same DCR, and all call this API at or near the same time.

Instead, use other DCR APIs that return only requested devices, such as `getDevices` and `getIdentityAttributes` (see recommendation 3, below). Most of these APIs require you to specify DCR Device IDs as arguments. Your application can retrieve the Device ID for DCR devices using the `getDeviceIdentifiers()` API call.

2. Observe a 5,000-device limit on API calls.

All DCR APIs that return an “Array of Device objects” or take as arguments an “Array of DeviceId objects” (such as `addDevices` and `updateDevices`) are subject to the potential performance problems and risk described in recommendation 1, above.

To avoid this, do not use these APIs for more than 5,000 devices in a single call. If your application needs to call the API for more than 5,000 devices, segment the list of devices and call the API multiple times. Since all applications have a wrapper around the DCR APIs, this logic should be fairly easy to implement.

This is especially important because performance and scaling for multiple applications making simultaneous DCR API calls can not be easily handled. Large add or update operations can block API calls from other applications. It is the responsibility of applications making such calls to yield control so other applications can effectively use the DCR APIs.

3. Use the `getIdentityAttributes()` API as much as possible.

Use this API in preference to `getDevices()` wherever possible. This API returns a device object that contains only the following attributes: `display_name`, `management_ip_address`, `host_name`, `domain_name`, `device_identity` (if it is an AUS device), `SysObjectId` and `mdf_type`. Most of the time, application needs for device information, especially for display purposes (such as in the Device Selector), are satisfied by this subset of attribute data. In addition, this API executes much more quickly. For example: To process 1,000 devices, `getDevices` takes approximately 10 seconds, while `getIdentityAttributes` takes about 500 milliseconds.

CISCO CONFIDENTIAL

Note that recommendation 2, above, also applies to `getIdentityAttributes` (that is, do not use it to fetch more than 5,000 devices at a time).

4. Use DCR Server status-check APIs.

Before making some API calls, you will want to ensure you are not generating useless traffic by checking that the DCR Server is actually running. DCR provides two APIs to check the status of the DCR Server:

- `isRunning` checks the status of the local DCR Server. Most of the time, you will want to use this call to check DCR Server status.
- `isMasterDCRRunning` checks the status of the Master DCR server in the domain.

5. Ensure you have sufficient JVM thread stack and heap allocations.

Large numbers of device objects require large amounts of memory. If your Java application will process large numbers of device objects on a regular basis, you should adjust the JVM thread and heap parameters accordingly. Based on tests, large groups of DCR device objects typically require the following memory spaces:

Objects	Memory
1,000	3.9MB
2,000	6.3MB
5,000	13.4MB
10,000	24.6MB
20,000	47.7MB
50,000	115MB

6. Call the DCRProxy close() API at the end.

The `DCRProxy` class provides a `close()` method to clean up and free resources, including any open DB connections, when you are finished using it. Make sure that your code calls the `close()` method on each proxy object once you are done using the proxy object.

7. Use the proper DCRProxy constructor.

You need to ensure that you use the proper `DCRProxy` object constructor: the default, or one that controls the loading of log4J categories for DCR and CSTM classes. For more information, see the [“Creating the DCRProxy Object”](#) section on page 14-29.



CISCO CONFIDENTIAL

CHAPTER 15

Using the Core Logging API

The Core Logging API (CLA) allows both the Core (the object repository) and MDCs (Multiple Device Controllers) to save logging messages as well as auditing messages. It also allows Core and other MDCs to log to the same file from both Java and C++, and to be accessed seamlessly from either language. MDCs also use CLA to write to logs wherever necessary.

The Core Logging API (CLA) is accessible from both Java and C++. The Core Client Registry (CCR) contains information for each MDC that pertains to accessibility of certain types of logs as well as auditing. It also contains location information for the logs. The CLA accesses the information in the CCR.

Log4cpp is used as the logging API and simple static methods are created for users of the CLA to call. The Log4cpp API remains hidden to users of the CLA. A very simple java JNI interface is created to call the static CLA methods.

The following topics describe the CLA:

- [About the Core Logging API Structure](#)
- [Using the Core Logging API](#)
- [About the Core Logging API Interface Design](#)

About the Core Logging API Structure

The Core Logging API consists of four parts:

1. **CoreLogger:** Communicates with the CCR to determine categories, priorities, and locations for MDC logs. Appropriate appenders are created for each MDC, depending on the location. They are attached to each category of the MDC with the specified priority. Priority levels are (ranked from lowest to highest):
 - DEBUG
 - INFO
 - WARNING
 - ERROR
 - FATALAUDIT priority is used to separate levels from each other.
2. **Logger API:** Contains a series of simple static methods, which CLA users call to create logs. It is the access point for the CLA into the Log4cpp API.

CISCO CONFIDENTIAL

3. **JavaLogger API:** Identical to the Logger API, but accessible from Java.
4. **Auditlog API:** A wrapper over JavaLogger which provides only the JavaLogger auditing feature. Your application can use this API if your code is compiled with JDK1.4 and above.

Using the Core Logging API

The following topics explain how to use each of the main CLA functions:

- [Initializing the Core Logging API](#)
- [Creating Debug Messages](#)
- [Creating Information Messages](#)
- [Creating Warning Messages](#)
- [Creating Error Messages](#)
- [Creating Fatal Messages](#)
- [Creating Auditing Messages](#)
- [Altering Priority for Category](#)
- [Adding Logging Location to CCR](#)
- [Adding Logging Category and Priority to CCR](#)

Initializing the Core Logging API

To initialize the CLA, use:

```
Call Logger::LoadCategories()
```

For Java:

```
call JavaLogger.LoadCategories()
```

Creating Debug Messages

The following information is required: MDC to which the message belongs, its category, the message, and its ID.

```
Call Logger::Debug(mdc, category, message, id)
```

For Java:

```
call JavaLogger.Debug(mdc, category, message, id)
```

Creating Information Messages

The following information is required: MDC to which the message belongs, its category, the message, and its ID.

```
Call Logger::Info(mdc, category, message, id)
```

CISCO CONFIDENTIAL

For Java:

```
call JavaLogger.Info(mdc, category, message, id)
```

Creating Warning Messages

The following information is required: MDC to which the message belongs, its category, the message, and its ID.

```
Call Logger::Warn(mdc, category, message, id)
```

For Java:

```
call JavaLogger.Warn(mdc, category, message, id)
```

Creating Error Messages

The following information is required: MDC to which the message belongs, its category, the message, and its ID.

```
Call Logger::Error(mdc, category, message, id)
```

For Java:

```
call JavaLogger.Error(mdc, category, message, id)
```

Creating Fatal Messages

The following information is required: MDC to which the message belongs, its category, the message, and its ID.

```
Call Logger::Fatal(mdc, category, message, id)
```

For Java:

```
call JavaLogger.Fatal(mdc, category, message, id)
```

Creating Auditing Messages

Auditing messages can be created using Java by using:

```
call JavaLogger.audit(userName, flag, taskid, appid, message)
```

Auditing functionality is not available from C++.

Altering Priority for Category

You can alter the priority level of a category within an MDC using:

```
Call Logger::AlterPriority(mdc, category, newPriority)
```

CISCO CONFIDENTIAL

For Java:

```
call JavaLogger.AlterPriority(mdc, category, newPriority)
```

Adding Logging Location to CCR

From a CCRInterface object, you can use:

```
call addLoggingCategory(mdc, category, priority)
```

From CCRAccess, you can use:

```
CCRAccess addLog mdc category priority
```

Adding Logging Category and Priority to CCR

From a CCRInterface object, you can use:

```
call setLoggingLocation(mdc, location)
```

From CCRAccess, you can use:

```
CCRAccess addLogLocation mdc location
```

About the Core Logging API Interface Design

The following topics describes the functions, fields, methods and arguments of the main Core Logging API components:

- [About the Logger Interface](#)
- [About the JavaLogger Interface](#)
- [About the Auditlog Interface](#)

About the Logger Interface

1. `public static void LoadCategories();`
Initializes the CLA.
2. `public static void AlterPriority(std::string mdc, std::string category, std::string priority);`
Alters a category's priority.

Parameter	Type	Purpose
mdc	std::string	The MDC to which the category belongs.
category	std::string	The category in the MDC whose priority will be altered.
priority	std::string	The new priority value.

CISCO CONFIDENTIAL

```
3. public static void Debug(std::string mdc, std::string category, std::string message,
    std::string id);
```

Posts a debug message if DEBUG priority is enabled.

Parameter	Type	Purpose
mdc	std::string	The MDC to which the message belongs.
category	std::string	The category in the MDC to which the message belongs.
message	std::string	The message.
id	std::string	The message ID.

```
4. public static void Info(std::string mdc, std::string category, std::string message,
    std::string id);
```

Posts an info message if INFO priority or less is enabled.

Parameter	Type	Purpose
mdc	std::string	The MDC to which the message belongs.
category	std::string	The category in the MDC to which the message belongs.
message	std::string	The message.
id	std::string	The message ID.

```
5. public static void Warn(std::string mdc, std::string category, std::string message,
    std::string id);
```

Posts a warning message if WARNING priority or less is enabled.

Parameter	Type	Purpose
mdc	std::string	The MDC to which the message belongs.
category	std::string	The category in the MDC to which the message belongs.
message	std::string	The message.
id	std::string	The message ID.

```
6. public static void Error(std::string mdc, std::string category, std::string message,
    std::string id);
```

Posts an error if ERROR priority or less is enabled.

Parameter	Type	Purpose
mdc	std::string	The MDC to which the message belongs.
category	std::string	The category in the MDC to which the message belongs.
message	std::string	The message.
id	std::string	The message ID.

CISCO CONFIDENTIAL

```
7. public static void Fatal(std::string mdc, std::string category, std::string message,
    std::string id);
```

Posts a fatal message if FATAL priority or less is enabled.

Parameter	Type	Purpose
mdc	std::string	The MDC to which the message belongs.
category	std::string	The category in the MDC to which the message belongs.
message	std::string	The message.
id	std::string	The message ID.

About the JavaLogger Interface

```
1. public static synchronized void LoadCategories();
```

Initializes the CLA.

```
2. public static synchronized void AlterPriority(string mdc, string category, string
    priority);
```

Alters a category's priority.

Parameter	Type	Purpose
mdc	string	The MDC to which the category belongs.
category	string	The category in the MDC whose priority will be altered.
priority	string	The new priority value.

```
3. public static synchronized void Debug(string mdc, string category, string message,
    string id);
```

Posts a debug message if DEBUG priority is enabled.

Parameter	Type	Purpose
mdc	string	The MDC to which the message belongs.
category	string	The category in the MDC to which the message belongs.
message	string	The message.
id	string	The message ID.

```
4. public static synchronized void Info(string mdc, string category, string message,
    string id);
```

Posts an info message if INFO priority or less is enabled.

Parameter	Type	Purpose
mdc	string	The MDC to which the message belongs.
category	string	The category in the MDC to which the message belongs.

CISCO CONFIDENTIAL

Parameter	Type	Purpose
message	string	The message.
id	string	The message ID.

5. `public static synchronized void Warn(string mdc, string category, string message, string id);`

Posts a warning message if WARNING priority or less is enabled.

Parameter	Type	Purpose
mdc	string	The MDC to which the message belongs.
category	string	The category in the MDC to which the message belongs.
message	string	The message.
id	string	The message ID.

6. `public static synchronized void Error(string mdc, string category, string message, string id);`

Posts an error message if ERROR priority or less is enabled.

Parameter	Type	Purpose
mdc	string	The MDC to which the message belongs.
category	string	The category in the MDC to which the message belongs.
message	string	The message.
id	string	The message ID.

7. `public static synchronized void Fatal(string mdc, string category, string message, string id);`

Posts a fatal message if FATAL priority or less is enabled.

Parameter	Type	Purpose
mdc	string	The MDC to which the message belongs.
category	string	The category in the MDC to which the message belongs.
message	string	The message.
Id	string	The message ID.

8. `public static synchronized void audit(string userName, byte flag, int taskid, string appid, string message)`

Sends an auditing event to be logged.

CISCO CONFIDENTIAL

Parameter	Type	Purpose
userName	string	The user name that posted the auditing message.
flag	byte	The flag of the event to be logged
taskid	int	The task ID.
appid	string	The application ID.
message	string	The auditing message.

9. `public static synchronized void audit(string userName, AccountParam params)`

Sends an auditing event to be logged

Parameter Name	Type	Purpose
userName	string	The user name that posted the auditing message.
params	AccountParam	The parameters of the audit log.

About the Auditlog Interface

1. `public static synchronized void audit(string userName, byte flag, int taskid, string appid, string message)`

Sends an auditing event to be logged. Auditlog.java is part of the com.cisco.core.nm.util package.

Parameter	Type	Purpose
userName	string	The user name that posted the auditing message.
flag	byte	The flag of the event to be logged
taskid	int	The task ID.
appid	string	The application ID.
message	string	The auditing message.



CISCO CONFIDENTIAL

CHAPTER 16

Adding Online Help

CWCS uses a customized Cisco help engine to display online help. The Cisco help engine produces a help system suite, composed of one or more help systems that you can install or uninstall when you need them. When you add a new application to CWCS:

- The help system for your application is added to the help system suite.
- Your application name appears in the main help contents page and index.
- The application name is linked to your help system's default page.

The following topics describe the CWCS help engine and how to add help for your application to the help system suite:

- [Overview of Online Help](#)
- [Implementing Help: Engineering Tasks](#)
- [Implementing Help: Writing Tasks](#)
- [Adding Drop-In Help Systems](#)

For more information about the Cisco help system, refer to the [Cisco Online Help support website \(http://www.in-olh-support.cisco.com/index.html\)](http://www.in-olh-support.cisco.com/index.html). This website provides access to templates, APIs, and many process documents, including:

- *EMBU Help System (Client/Web Server) 2.0 System Functional Specification*, EDCS-126498
- *Online Help System for CMF 2.3 System Functional Specification*, EDCS-238093
- *Creating Cisco Online Help Using FrameMaker and WebWorks*, ENG-104742
- *Creating Online Help Using Native HTML*, EDCS-298882
- *Creating Online Help Using XML*, EDCS-357012
- *Guidelines for Writing Single-Sourced Manuals*, ENG-70452
- *Editing Guidelines for Writing Documentation*, EDCS-280677



Note

The Cisco help system no longer supports authoring in RoboHelp. Authors creating online help for CiscoWorks applications should convert existing RoboHelp projects to native HTML, using the methods detailed in EDCS-298882.

This release of CWCS uses Cisco help system version 2.0, which incorporates the following major changes from version 1.0:

CISCO CONFIDENTIAL

Overview of Online Help

The major changes for Online Help since Version 1.0 includes the following:

- The help system supports the Cisco UE/UII look and feel. This includes access to PDF files and Glossary topics as topics within the help system, rather than links from banner buttons.
- The table of contents and index display using HTML and Javascript only.
- The banner navigation elements display using HTML tables instead of gif files.

Cisco help consists of:

- Help window layouts defined by several special HTML files.
- A help engine that:
 - Links context-sensitive help buttons to the appropriate topic.
 - Creates a master table of contents and index for the installed help systems.
 - Provides information required by the search engine.
 - Handles error conditions.
- A search engine and search index files that support full-text searches across one or all installed help systems.
- Special mapping files that match help tags in your application to help topics.

The following topics describe these components:

- [How Help Is Displayed](#)
- [Understanding the Help Engine](#)
- [Understanding the Search Engine](#)
- [Understanding Mapping Files](#)

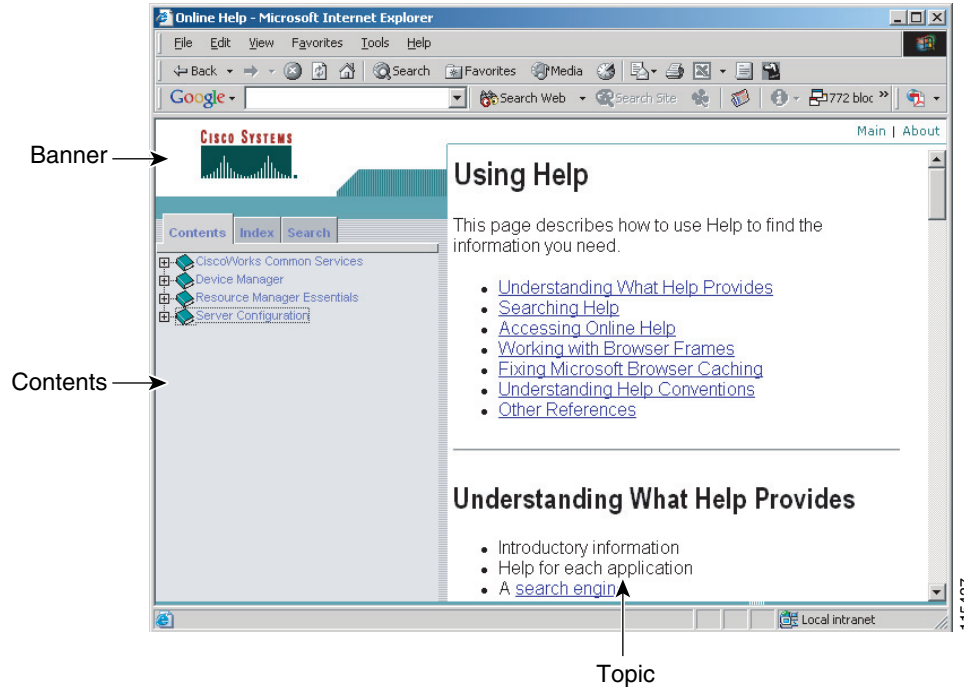
How Help Is Displayed

The Cisco help engine displays help files in a separate browser window.

To organize the content of a suite of help systems, the help engine creates a top-level help page that contains links to the help system for each installed application. [Figure 16-1](#) shows an example of the Main Help Window for a help system suite.

CISCO CONFIDENTIAL

Figure 16-1 Help Suite: Main Page Layout



For this top-level page in a help suite, the help window contents and functions are as follows:

- The **Banner** frame contains two navigation links on the far right:
 - The **Main** link reloads the master table of contents page with its associated default topic, closing any open folders in the contents tree.
- The **Contents** frame provides access to the contents, index and search functions for the help system suite. The Contents frame contains three tabs:
 - The **Contents** tab contains entries for all installed help systems, in expanding and collapsing folders (represented by book icons). The contents are organized by product suite or other groups. This organization helps users quickly find the help for an application in the suite.
 - The **Index** tab contains a link to the first index page of each installed help system. These entries are organized alphabetically, by application name. Each application index entry takes you to an alphabetized list of links to all the topics in the selected help system.
 - The **Search** tab loads a search page in the Topic frame. You can search the individual help system or all the help systems in a help system suite.
- The **Topic** frame contains the actual help information. The default topic for the master help system is “Using Help.”

These frames are created by a file called `index.html`. For a help system suite, there is one `index.html` file *per help system* as well as a master, or main, `index.html` file.

The look of individual pages with only one help system is nearly identical to [Figure 16-1](#). The only differences are:

- The **Contents** tab contains entries only for the single help system topics.
- The **Index** tab contains alphabetized links to all the topics in the help system, rather than to the first page of the index for each help system.

CISCO CONFIDENTIAL

Understanding the Help Engine

The Cisco help engine is a customized Java application that displays help topics in a separate browser window. This help engine:

- Displays help topics associated with the navigation tree entries in the CWCS desktop
- Links context-sensitive help buttons to the appropriate topic
- Creates a master table of contents and index for all installed help systems
- Handles error conditions

The following topics describe these help engine functions:

- [Displaying Help Topics](#)
- [Linking Context-Sensitive Help Buttons](#)
- [Creating the Main Help Contents and Index](#)
- [Handling Error Conditions](#)
- [Summarizing the Display Process](#)

Displaying Help Topics

The Cisco help engine uses the following behaviors to display help topics associated with the navigation tree options in the CWCS desktop:

- If the user clicks the desktop Help button and no folder or task is selected, the help engine displays the Main Help page (see the [“How Help Is Displayed”](#) section on page 16-2). This page displays the “Using Help” page in the Topics frame and a list of all installed help systems in the Contents frame.
- If the user clicks the desktop Help button and selects a folder in the navigation tree, the help engine displays the first page for that help system (usually the overview) in the Topics frame. The Contents frame contains the contents for that application’s help system.

The first page of a help system also appears when you select Help from the toolbar (where the application has toolbar), then chooses the option for this application.

- If the user selects a *task* in the navigation tree, the help engine displays the help page for that task in the Topics frame. The Contents frame contains the contents for that application’s help system.

Linking Context-Sensitive Help Buttons

To support context-sensitive help, each dialog box requires a link to its associated help topic. Therefore, for each dialog box, the engineer must add an engineering tag to the code. Mapping files correlate these engineering tags to corresponding help file paths. Mapping files are described in the [“Understanding Mapping Files”](#) section on page 16-9.



Note Each application, regardless of whether the writer creates the help topics using a Native HTML project or a FrameMaker file, must have a separate mapping file. These mapping files must be located in a special directory at run time (see the [“Packaging the Help Files”](#) section on page 16-16).

When the user opens the help system from a context-sensitive link, such as a Help button on a dialog box, the help engine creates a URL to display the topic specified in the mapping file.

CISCO CONFIDENTIAL

Creating the Main Help Contents and Index

The Cisco help engine creates the master table of contents and index files for the installed help systems.

The help engine reads special DROPIN and DROPINPLACE lines in each mapping file. Using this information, it creates a top-level help page that contains links to the help files for all installed applications, including in-house and third-party drop-ins. If no DROPIN or DROPINPLACE lines are present in any mapping files, the help system does not create the top-level help page.

Related Topics

- [Understanding Mapping Files](#)
- [Adding Drop-In Help Systems](#)

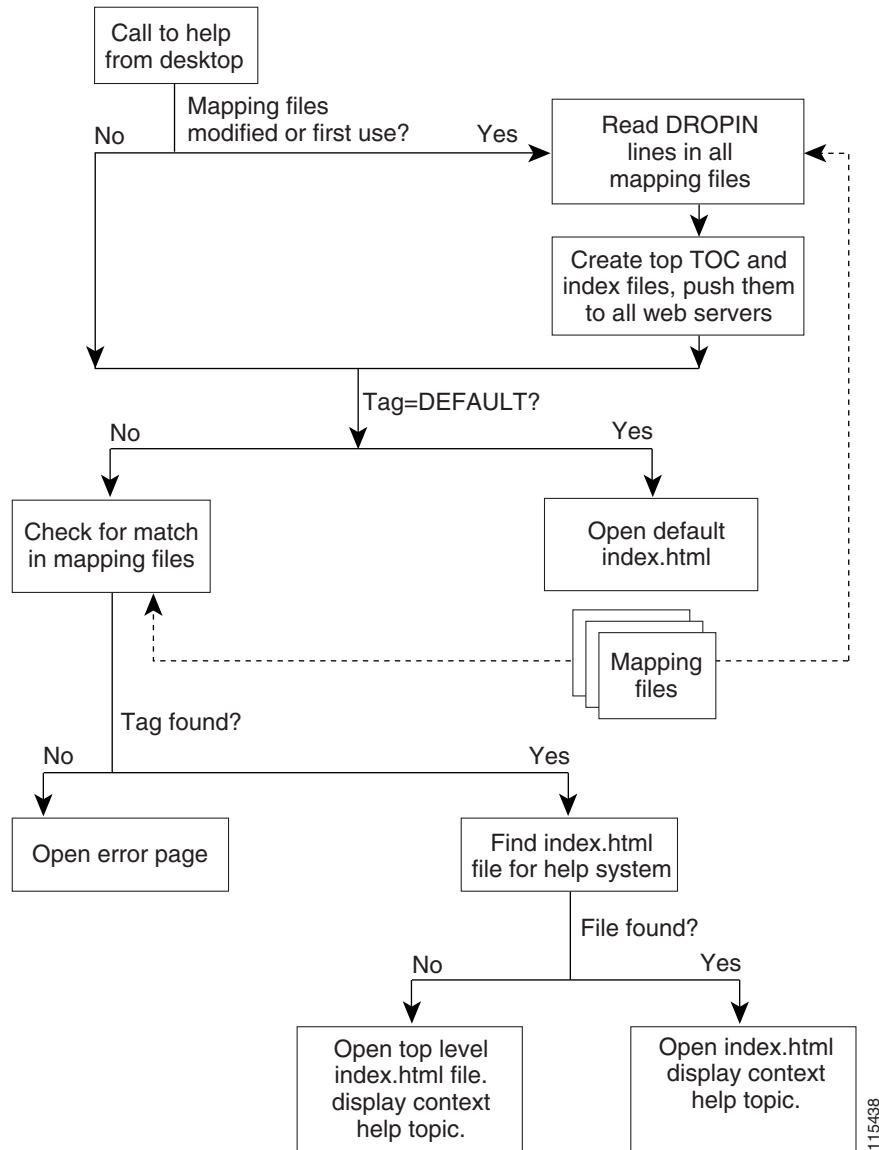
Handling Error Conditions

The Cisco help engine handles context-sensitive links when the target file is not available. The Cisco help engine reads all the mapping files, looking for a match to the engineering tag sent by the application.

- If no match for the engineering tag is found, the Cisco help engine loads a standard error page in the Topic frame.
- If a match is found for the engineering tag, but the index.html file for the help system is missing, the engine displays the help topic anyway, using a default index.html file and the help topic filepath.

Summarizing the Display Process

Figure 16-2 illustrates the process the Cisco help engine API uses to display the help topics.

CISCO CONFIDENTIAL**Figure 16-2 Help API Process Overview**

Understanding the Search Engine

The search engine allows the user to search all files in the help system or the help files for a single help system. The search engine consists of several parts, discussed in the following topics:

- [Displaying the Search Dialog Box](#)
- [Searching the Files and Displaying the Search Results](#)
- [Summarizing the Search Process](#)

CISCO CONFIDENTIAL

Displaying the Search Dialog Box

When the reader clicks the Search button in the banner frame, HelpSearchServlet.java displays the search dialog box in the topic frame. [Figure 16-3](#) shows a sample search dialog box.

Figure 16-3 *Sample Search Dialog Box*



Mapping files define the scope of the search for each help system. A special SEARCH line in each mapping file defines:

- The search scope list displayed in the search page. For example, in [Figure 16-3](#), the search scope is “CiscoWorks Common Services”.
- The search index files to be searched when the user selects that entry. The search engine reads the search index files instead of opening, reading, and closing every topic file in a help system. Each line in the search index file contains the text in one topic file.



Note Each application must have a separate search index file. These search index files must be located in a special directory at run time (see the [“Packaging the Help Files”](#) section on page 16-16).

Related Topics

- [Understanding Mapping Files](#)
- [Maintaining the Search Index File](#)

Searching the Files and Displaying the Search Results

After the user has entered the search parameters, HelpSearchServlet.java looks for matches in the search index files and displays the results. [Figure 16-4](#) shows a sample search results page. When the user clicks a link in the search results page, the search engine displays the associated help topic.

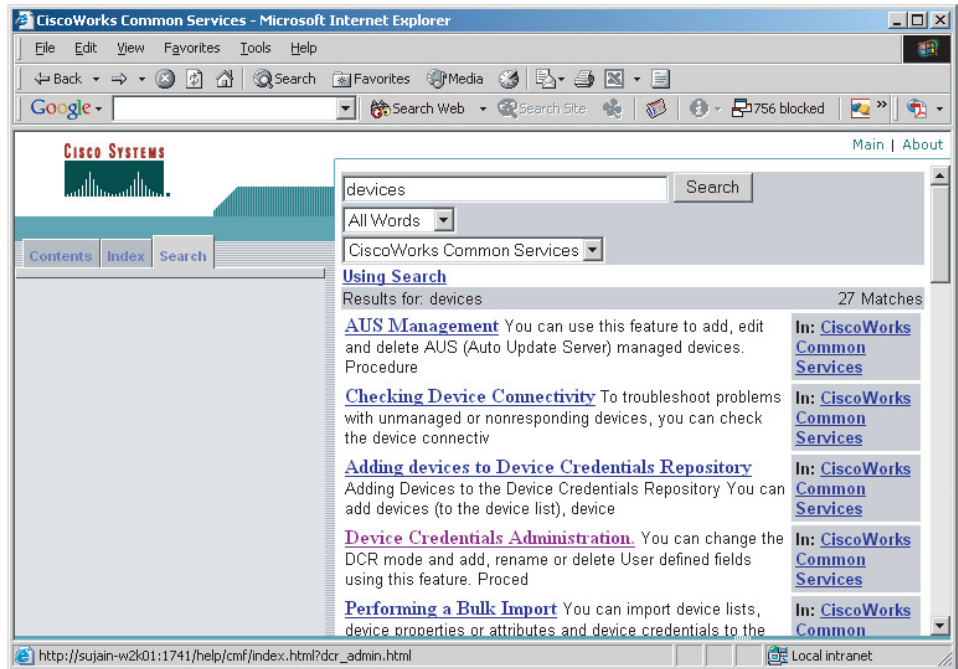
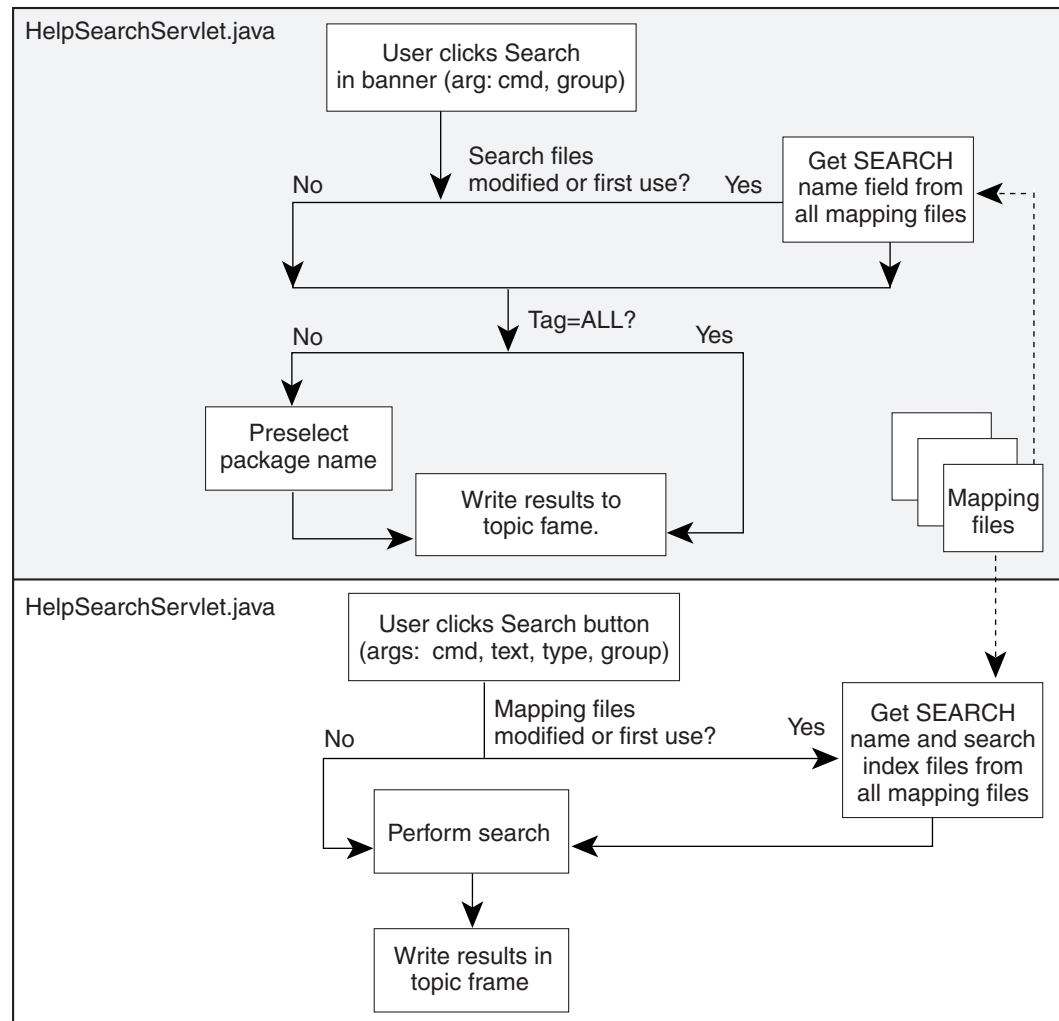
CISCO CONFIDENTIAL**Figure 16-4** Sample Search Results Page**Summarizing the Search Process**

Figure 16-5 illustrates the process the search engine uses to find and display its results.

CISCO CONFIDENTIAL**Figure 16-5 Search Process Overview**

115441

Understanding Mapping Files

Mapping files are an integral part of the help display process. The Cisco help engine API uses mapping files to:

- Display help topics associated with navigation tree entries in the desktop
- Link context-sensitive help buttons to the appropriate topic
- Create a master table of contents and index for the help system suite

The search engine also depends on mapping files to define:

- The entry in the search scope list that is displayed in the search dialog box
- The search index files to be searched when the user selects that entry

The following topics contain more information about mapping files:

- [Mapping File Conventions and Requirements](#)

CISCO CONFIDENTIAL

- [Mapping File Line Types](#)
- [Sample Mapping File](#)

Mapping File Conventions and Requirements

The conventions shown in [Table 16-1](#) must be used by all mapping files for the Cisco help engine API to work properly:

Table 16-1 Mapping File Conventions

File Name	<p>Because all mapping files are installed in a single directory at run time, each mapping file must have a unique name. Use the following naming convention:</p> <pre>productSubsystem.hlp</pre> <p>where:</p> <ul style="list-style-type: none"> • <i>product</i> is the product acronym • <i>Subsystem</i> is the subsystem name or acronym • <i>.hlp</i> is the required file extension <p>Example: CRMCmf.hlp</p>
Runtime Location	<p>The Cisco help engine looks for mapping files in the following runtime location:</p> <pre>NMSROOT/htdocs/help/mappingfiles</pre> <p>where <i>NMSROOT</i> is the directory in which the product was installed.</p>
Guidelines	<p>All mapping files must follow these conventions:</p> <ul style="list-style-type: none"> • Each tag entry must be contained within one line. • Tags must be unique across <i>all</i> installed mapping files. • The tag and filepath must be separated by at least one blank space. • The filepath must include all directories under /help; the /help directory is assumed by the Cisco help engine. • The # indicates a comment line. • Level names and order# fields must be contained within double quotes. • Level names and order# fields cannot contain embedded double quotes. • Filepath must be preceded by a backslash (/). The top-level “/help” directory is assumed. • DROPIN, DROPINPLACE, SEARCH, and SEARCHALL keywords must be left-justified. • If the application is available from the desktop navigation bar, the engineer must add the help tag to the application registry file. • At run time all mapping files must be installed in the ../help/mappingfiles directory.

Mapping File Line Types

[Table 16-2](#) summarizes the available mapping file line types. These line types allow you to:

- Implement context-sensitive help
- Add an entry to the main help contents
- Determine the order of the entries in the main help page contents

CISCO CONFIDENTIAL

- Define the entry in the search scope list
- Define the search index files to be searched
- Define the text for searching “all” help systems

Table 16-2 Mapping File Line Types

Line Type	Description
Tag/helpfile	<p>The tag/filepath pair used to implement context-sensitive help.</p> <pre>tag /filepath</pre> <p>where:</p> <p><i>tag</i> is a help tag name. It must be unique among all installed mapping files.</p> <p><i>/filepath</i> is the path name of the HTML file. The Cisco help engine assumes the top-level directory is /help.</p>
Comment	<p>Refers to any comment or description</p> <pre># <any text></pre>
DROPIN	<p>Adds an entry to the contents in the main help page.</p> <pre>DROPIN, "level 1 name" [, "level x name"] [...] [, /filepath]</pre> <p>where:</p> <p><i>level x name</i> defines a contents structure (books and pages; books can contain other books or pages)</p> <p><i>/filepath</i> is the path name of the HTML file that corresponds to the last entry in the list. If <i>/filepath</i> is not present, <i>level x name</i> is added to the contents but no link is created.</p> <p>The number of <i>level x name</i> entries in the DROPIN line determines the location of the link in the contents hierarchy. Use DROPIN lines to add a link in the main help page contents to your help system when the order in which the help system entry appears in the list of book entries <i>is not important</i>.</p> <p>You can put DROPIN lines anywhere in the file, but it is best to put them before any tag/filepath pairs.</p>
DROPINPLACE	<p>Forces the last <i>level x name</i> entry to the top or end of the list in the main help page contents.</p> <pre>DROPINPLACE, "order#", "level 1 name" [, "level x name"] [...] [, /filepath]</pre> <p>where:</p> <p><i>order#</i> is the order in which these entries appear. For details on how the help system uses this value to order the main help page contents, see the “Defining the Main Help Page Contents and Index” section on page 16-24.</p> <p><i>level x name</i> defines a contents structure.</p> <p><i>/filepath</i> is the path name of the HTML file that corresponds to the last entry in the list. If <i>/filepath</i> is not present, <i>level x name</i> is added to the contents but no link is created.</p> <p>Use DROPINPLACE lines to force book or page entries for a help system to appear in a specific location in that book in relation to other entries at the same level.</p> <p>You can put DROPINPLACE lines anywhere in the file, but it is best to put them at the top, before any tag/filepath pairs, for easy access.</p>

CISCO CONFIDENTIAL**Table 16-2 Mapping File Line Types (continued)**

Line Type	Description
SEARCH	<p>Defines the entry in the search scope list that is displayed in the search page and the search index files to be searched when the user selects that entry.</p> <p><i>SEARCH, "search scope", "app.sch"[, "app.sch"...]</i></p> <p>where:</p> <p><i>search scope</i> is the name you want to appear in the search drop-down box. If this field is not present, the search results page displays "Unknown" in the application name column for any hits to this help system.</p> <p><i>app.sch</i> is the name of the search index files you want the search utility to look at when the user selects this option. You can specify multiple search files. For example, if you want to search your help topics and the glossary, make sure there is a search index for the glossary, then add that filename to this line. For example:</p> <p><i>SEARCH, "Management Connection", "mngconnect.sch", "glossary.sch"</i></p>
SEARCHALL	<p>Special line that defines the text to search "all" help systems.</p> <p><i>SEARCHALL "all_name"</i></p> <p>If SEARCHALL is not specified in any installed mapping file, the search engine uses the default text "All."</p>

Sample Mapping File

[Example 16-1](#) shows a typical mapping file.

Example 16-1 Sample Mapping File

```
#####
# MyApp Manager Mapping file (MyAppmgr.hlp)
# for context-sensitive help.
#
# Last modified 02/31/2004
#
# Copyright (c) 1997-2004 by Cisco Systems, Inc.
# All rights reserved
#####
#####
# Top-level TOC entry
#####
DROPIN, "Server Configuration", "Administration", "Process Management", /myappmgr/index.html
SEARCH, "Process Management", "myappmgr.sch"

#-----
# TOP-LEVEL FOLDER
#-----
# This is for the MyApp Server > Administration > Process Management
procmgr_folder /myappmgr/myappmgr_overv.html

#-----
# PROCESS ADMIN OPERATIONS
#-----
# this page would start a single process or system
myapp_mgr_start_processes /myappmgr/ad_procs_start.html
```


CISCO CONFIDENTIAL

```
# this page would stop a single process or system
myapp_mgr_stop_processes /myappmgr/ad_procs_stop.html

# this page would display myapp.log with selected process
myapp_mgr_myapp_log

# this page would display the status of processes
myapp_mgr_process_status /myappmgr/ad_procs_status.html

# this page would display process properties
myapp_mgr_process_report /myappmgr/ad_procs_detail.html

#----- EOF -----#
```

Implementing Help: Engineering Tasks

Implementing help for a new application typically requires effort on the part of both the development engineers and the writers. These tasks must be performed by the engineer:

1. **Installing the Help Packages**—The online help and search engine class files must be installed in the classpath. You can do this by installing the appropriate CWCS help engine packages.
2. **Adding a Call to the Help Engine**—Each code module that creates a dialog box with a Help button must include a call to the Cisco help engine servlet.
3. **Updating the Mapping File**—The engineer must help the writer keep the mapping file current.
4. **Packaging the Help Files**—For the Cisco help engine API to work, help files must be installed in predetermined locations in the runtime environment.

Installing the Help Packages

CWCS supplies the help and search engine class files as part of the pxhlp (Windows) and CSCOhlp (Solaris) packages. To install the Help system, include these packages in your install. If performed correctly, the following files should appear in your classpath:

```
com\cisco\nm\help\ClientServerHelpAPI.class
com\cisco\nm\help\ConfigReadWrite.class
com\cisco\nm\help\CvHelpApiIf.class
com\cisco\nm\help\DataHandler.class
com\cisco\nm\help\HelpCache.class
com\cisco\nm\help\HelpCacheServlet.class
com\cisco\nm\help\HelpConstants.class
com\cisco\nm\help\HelpEngine.class
com\cisco\nm\help\HelpSearch.class
com\cisco\nm\help\HelpSearchServlet.class
com\cisco\nm\help\HelpTree.class
com\cisco\nm\help\HelpTreeNode.class
com\cisco\nm\help\ListenerServlet.class
com\cisco\nm\help\PopHelp.class
com\cisco\nm\help\PostSearch.class
com\cisco\nm\help\PreSearch.class
com\cisco\nm\help\SearchHelpServlet.class
com\cisco\nm\help\ServerHelpEngine.class
com\cisco\nm\help\ServerSearchEngine.class
com\cisco\nm\help\SystemUtils.class
```

CISCO CONFIDENTIAL

For a complete list of required files, see the pxhlp.bom file in the ClearCase VOB for the version of CWCS you are using.

Adding a Call to the Help Engine

The following topics describe how to call the Cisco help engine:

- [Calling Help From a Java Application](#)
- [Calling Help From an HTML-Based Application](#)

Calling Help From a Java Application

Use the Java class `ClientServerHelpAPI` summarized in [Table 16-3](#) to start the help system from your Java application. Each code module that creates a dialog box with a Help button must include a call to this servlet.

Table 16-3 *ClientServerHelpAPI Summary*

Name	ClientServerHelpAPI	
Description	Wraps the Cisco help engine call into a convenient API. This version starts the help system only from a web server-based Java applet. Starts the help system only if the applet passed is <i>not null</i> and the parameter is valid. This API does not work when run as a “file://” call.	
Parameters	Syntax	Action
	TAG= <i>some_help_tag</i>	Opens a browser window and displays the correct help context.
	TAG=DEFAULT	Displays the help system’s top-level table of contents.
	URL= <i>some_url</i>	Opens a browser window and displays the URL “ <i>some_url</i> ”, where <i>some_url</i> is a complete valid URL. (See <code>java.net.URL</code> for information on URL formatting.) Typically used to display third-party help systems.
	All other parameters	Throws an exception and does not display the help system.
Example	<p>For each code module that creates a dialog box with a Help button, add the import statement and the call to the servlet:</p> <pre>import com.cisco.nm.help.ClientServerHelpAPI; ... public class MyApplet extends Applet { ... public handleHelp(...) { ... try{ ClientServerHelpAPI.launchHelp(getApplet(), "<parameter>"); }catch(MalformedURLException e){} catch(Exception e){} ... } ... }</pre>	

CISCO CONFIDENTIAL**Calling Help From an HTML-Based Application**

Use the JavaScript function `pophelp` to start the help system from your HTML-based application. This function exists in the parent window (the CWCS desktop). Each HTML dialog box that includes a help button must include a call to this function.

Table 16-4 Pophelp Function Summary

Name	pophelp	
Description	Wraps the Cisco help engine call into a convenient API. This version starts the help system from a web server-based HTML application. This API does not work when run a “file://” call.	
Parameters	Syntax	Action
	TAG= <i>some_help_tag</i>	Opens a browser window and displays the correct help context. If no help tag is passed, displays the top-level table of contents for the help system.
	TAG=DEFAULT	Displays the help system’s top-level table of contents.
	All other parameters	Displays the help system’s top-level table of contents.
Example	(all on one line): <pre><input type="button" name="_Help" value="Help" onclick="parent.pophelp('TAG=some-help-tag');"></pre>	
Function Listing	If you cannot access the CWCS desktop version, add this code to your HTML file. <i>Note that the path /CSCOnm? may change depending on how your web server is configured.</i> <pre><SCRIPT LANGUAGE="JavaScript"> <!-- function pophelp(tag) { if(!tag) { tag = "DEFAULT"; } window.open("/CSCOnm/servlet/com.cisco.nm.help.ServerHelpEngine?tag=" + tag, "HelpSystem", "toolbar=yes,location=no,directories=no,status=yes,menubar=yes,resizable=yes,scrollbars=yes,width=700,height=575"); } //--> </SCRIPT></pre>	

Updating the Mapping File

To implement context-sensitive help, the engineer must work closely with the help writer. Although the writer typically creates and maintains the mapping file, the engineer is responsible for:

- Inserting the engineering tags into the source code.
- Adding the engineering tags to the application registry files.
- Notifying the writer of any new dialog boxes. Each new dialog box requires adding an engineering tag/filepath pair to the mapping file for that application.
- Notify the writer of any changes to existing tags (for example, deleted dialog boxes).

**Note**

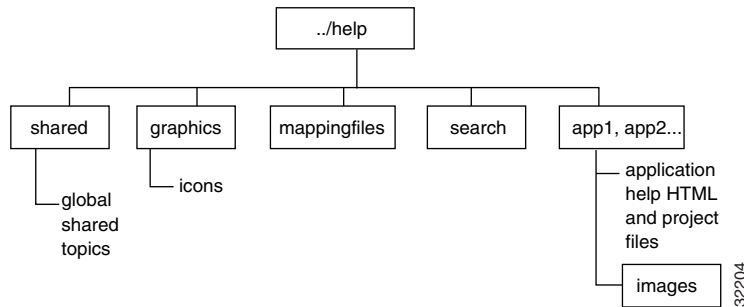
All engineering tags must follow the conventions described in the “[Mapping File Conventions and Requirements](#)” section on page 16-10.

CISCO CONFIDENTIAL

Packaging the Help Files

The required runtime structure, shown in [Figure 16-6](#), includes all help systems and any shared help resources.

Figure 16-6 Required Help Runtime Structure



[Table 16-5](#) describes each directory and lists the required files that should be installed in each directory. For a complete list of required files, see the pxhlp.bom file in the ClearCase VOB for the version of CWCS you are using.

Table 16-5 Help System Runtime Structure

Directory	Description, Path, and Required Files										
Help root directory	<p>The top level of the help structure.</p> <p>Path: <i>NMSROOT</i>/htdocs/help, where <i>NMSROOT</i> is the directory in which the product was installed.</p> <p>Required files:</p> <table> <tbody> <tr> <td>config.js</td> <td>nav.html</td> </tr> <tr> <td>errorfile.html</td> <td>pophelp.html</td> </tr> <tr> <td>helpconfig.txt</td> <td>toc.js</td> </tr> <tr> <td>ind.js</td> <td>toolbar.html</td> </tr> <tr> <td>index.html</td> <td>utils.js</td> </tr> </tbody> </table>	config.js	nav.html	errorfile.html	pophelp.html	helpconfig.txt	toc.js	ind.js	toolbar.html	index.html	utils.js
config.js	nav.html										
errorfile.html	pophelp.html										
helpconfig.txt	toc.js										
ind.js	toolbar.html										
index.html	utils.js										

CISCO CONFIDENTIAL**Table 16-5 Help System Runtime Structure (continued)**

Directory	Description, Path, and Required Files																																																						
shared	<p>Contains all resources that are shared globally across all help systems.</p> <p>Path: <i>NMSROOT</i>/htdocs/help/shared</p> <p>Required files:</p> <table border="0"> <tr> <td>ban.html</td> <td>more_info.html</td> </tr> <tr> <td>banner.html</td> <td>nav.html</td> </tr> <tr> <td>before_login.html</td> <td>pdf_gloss.js</td> </tr> <tr> <td>cco_login.html</td> <td>ref_help.html</td> </tr> <tr> <td>content.css</td> <td>searchframe.html</td> </tr> <tr> <td>content_ns.css</td> <td>sel_dev.html</td> </tr> <tr> <td>cw2000.html</td> <td>sel_opt.html</td> </tr> <tr> <td>footer.html</td> <td>sel_url.html</td> </tr> <tr> <td>global.css</td> <td>shared.hhc</td> </tr> <tr> <td>global_ns.css</td> <td>shared.hhk</td> </tr> <tr> <td>global_ns_sol.css</td> <td>sp.html</td> </tr> <tr> <td>help_tips.html</td> <td>style_1.css</td> </tr> <tr> <td>ind.html</td> <td>sync.js</td> </tr> <tr> <td>index.html</td> <td>toolbar.html</td> </tr> <tr> <td>indexframe.html</td> <td>tree.js</td> </tr> <tr> <td>leftframe.html</td> <td>utils.js</td> </tr> <tr> <td>login.html</td> <td>WebHelp.jar</td> </tr> <tr> <td>menu.html</td> <td>WHIEInd.htm</td> </tr> <tr> <td>menu.js</td> <td>WHIEToc.htm</td> </tr> <tr> <td></td> <td>WHnonIE4.css</td> </tr> </table>	ban.html	more_info.html	banner.html	nav.html	before_login.html	pdf_gloss.js	cco_login.html	ref_help.html	content.css	searchframe.html	content_ns.css	sel_dev.html	cw2000.html	sel_opt.html	footer.html	sel_url.html	global.css	shared.hhc	global_ns.css	shared.hhk	global_ns_sol.css	sp.html	help_tips.html	style_1.css	ind.html	sync.js	index.html	toolbar.html	indexframe.html	tree.js	leftframe.html	utils.js	login.html	WebHelp.jar	menu.html	WHIEInd.htm	menu.js	WHIEToc.htm		WHnonIE4.css														
ban.html	more_info.html																																																						
banner.html	nav.html																																																						
before_login.html	pdf_gloss.js																																																						
cco_login.html	ref_help.html																																																						
content.css	searchframe.html																																																						
content_ns.css	sel_dev.html																																																						
cw2000.html	sel_opt.html																																																						
footer.html	sel_url.html																																																						
global.css	shared.hhc																																																						
global_ns.css	shared.hhk																																																						
global_ns_sol.css	sp.html																																																						
help_tips.html	style_1.css																																																						
ind.html	sync.js																																																						
index.html	toolbar.html																																																						
indexframe.html	tree.js																																																						
leftframe.html	utils.js																																																						
login.html	WebHelp.jar																																																						
menu.html	WHIEInd.htm																																																						
menu.js	WHIEToc.htm																																																						
	WHnonIE4.css																																																						
graphics	<p>Contains graphics that are shared globally across all the products.</p> <p>Path: <i>NMSROOT</i>/htdocs/help/graphics</p> <p>Required files:</p> <table border="0"> <tr> <td>Admin.gif</td> <td>hlp_bg.gif</td> </tr> <tr> <td>caution.gif</td> <td>hlp_bg_nml.gif</td> </tr> <tr> <td>CiscoLogo.gif</td> <td>hlp_bg_sel.gif</td> </tr> <tr> <td>footer_left.gif</td> <td>hlp_bg_lt.gif</td> </tr> <tr> <td>gloss.gif</td> <td>hlp_bg_lt_nml.gif</td> </tr> <tr> <td>gloss_over.gif</td> <td>hlp_bg_lt_sel.gif</td> </tr> <tr> <td>help.gif</td> <td>hlp_bg_rt.gif</td> </tr> <tr> <td>help_over.gif</td> <td>hlp_bg_rt_nml.gif</td> </tr> <tr> <td>main.gif</td> <td>hlp_bg_rt_sel.gif</td> </tr> <tr> <td>main_over.gif</td> <td>img_vert_green.gif</td> </tr> <tr> <td>Operators.gif</td> <td>join.gif</td> </tr> <tr> <td>pdf.gif</td> <td>joinbottom.gif</td> </tr> <tr> <td>pdf_over.gif</td> <td>jointop.gif</td> </tr> <tr> <td>search.gif</td> <td>lgo_cisco.gif</td> </tr> <tr> <td>search_over.gif</td> <td>line.gif</td> </tr> <tr> <td>bg_vert_dash.gif</td> <td>minus.gif</td> </tr> <tr> <td>blank.gif</td> <td>minusbottom.gif</td> </tr> <tr> <td>blank_minus.gif</td> <td>minsonly.gif</td> </tr> <tr> <td>blank_plus.gif</td> <td>minustop.gif</td> </tr> <tr> <td>book_c_sel.gif</td> <td>plus.gif</td> </tr> <tr> <td>book_closed.gif</td> <td>plusbottom.gif</td> </tr> <tr> <td>book_o_sel.gif</td> <td>plusonly.gif</td> </tr> <tr> <td>book_open.gif</td> <td>plustop.gif</td> </tr> <tr> <td>footer_left.gif</td> <td>spacer.gif</td> </tr> <tr> <td>help_doc.gif</td> <td>vert_top.gif</td> </tr> <tr> <td>help_doc_mo.gif</td> <td></td> </tr> <tr> <td>help_doc_sel.gif</td> <td></td> </tr> </table>	Admin.gif	hlp_bg.gif	caution.gif	hlp_bg_nml.gif	CiscoLogo.gif	hlp_bg_sel.gif	footer_left.gif	hlp_bg_lt.gif	gloss.gif	hlp_bg_lt_nml.gif	gloss_over.gif	hlp_bg_lt_sel.gif	help.gif	hlp_bg_rt.gif	help_over.gif	hlp_bg_rt_nml.gif	main.gif	hlp_bg_rt_sel.gif	main_over.gif	img_vert_green.gif	Operators.gif	join.gif	pdf.gif	joinbottom.gif	pdf_over.gif	jointop.gif	search.gif	lgo_cisco.gif	search_over.gif	line.gif	bg_vert_dash.gif	minus.gif	blank.gif	minusbottom.gif	blank_minus.gif	minsonly.gif	blank_plus.gif	minustop.gif	book_c_sel.gif	plus.gif	book_closed.gif	plusbottom.gif	book_o_sel.gif	plusonly.gif	book_open.gif	plustop.gif	footer_left.gif	spacer.gif	help_doc.gif	vert_top.gif	help_doc_mo.gif		help_doc_sel.gif	
Admin.gif	hlp_bg.gif																																																						
caution.gif	hlp_bg_nml.gif																																																						
CiscoLogo.gif	hlp_bg_sel.gif																																																						
footer_left.gif	hlp_bg_lt.gif																																																						
gloss.gif	hlp_bg_lt_nml.gif																																																						
gloss_over.gif	hlp_bg_lt_sel.gif																																																						
help.gif	hlp_bg_rt.gif																																																						
help_over.gif	hlp_bg_rt_nml.gif																																																						
main.gif	hlp_bg_rt_sel.gif																																																						
main_over.gif	img_vert_green.gif																																																						
Operators.gif	join.gif																																																						
pdf.gif	joinbottom.gif																																																						
pdf_over.gif	jointop.gif																																																						
search.gif	lgo_cisco.gif																																																						
search_over.gif	line.gif																																																						
bg_vert_dash.gif	minus.gif																																																						
blank.gif	minusbottom.gif																																																						
blank_minus.gif	minsonly.gif																																																						
blank_plus.gif	minustop.gif																																																						
book_c_sel.gif	plus.gif																																																						
book_closed.gif	plusbottom.gif																																																						
book_o_sel.gif	plusonly.gif																																																						
book_open.gif	plustop.gif																																																						
footer_left.gif	spacer.gif																																																						
help_doc.gif	vert_top.gif																																																						
help_doc_mo.gif																																																							
help_doc_sel.gif																																																							

CISCO CONFIDENTIAL**Table 16-5** Help System Runtime Structure (continued)

Directory	Description, Path, and Required Files						
mappingfiles	Contains the mapping files for all installed help packages. Path: <i>NMSROOT</i> /htdocs/help/mappingfiles Additional required files: shared.hlp						
search	Contains the search index files for all installed help packages. Path: <i>NMSROOT</i> /htdocs/help/search Additional required files: <table border="0" style="width: 100%;"> <tr> <td style="width: 50%;">CMFshared.sch</td> <td style="width: 50%;">search.sch</td> </tr> <tr> <td>PreSearch.html</td> <td>SearchTemplate.html</td> </tr> <tr> <td>search.html</td> <td>searchtips.html</td> </tr> </table>	CMFshared.sch	search.sch	PreSearch.html	SearchTemplate.html	search.html	searchtips.html
CMFshared.sch	search.sch						
PreSearch.html	SearchTemplate.html						
search.html	searchtips.html						
app1, app2, ...	Contains the help files for each application. Any images that are only referenced by this application are stored in the images subdirectory for that application. Path: <i>NMSROOT</i> /htdocs/help/ <i>appname</i> Notes: <ul style="list-style-type: none"> • Be sure to install and uninstall the help for the application or device package with the application or device. • If you are installing an upgrade, there may not be a one-to-one correspondence to the old help files. Be sure your install program deletes the target directory first to ensure that no old files are left on the system. 						

Implementing Help: Writing Tasks

Implementing and maintaining online help typically requires that the writer perform these tasks:

-
- Step 1** [Selecting an Authoring Tool](#)—Cisco help was designed to allow writers to use any native HTML editor (such as HTML Help Workshop), FrameMaker/Webworks, or an XML editor (like XMetal) as their authoring application.
 - Step 2** [Setting Up Your Authoring Environment](#)—Using a directory structure that closely resembles the runtime directory will make testing and delivering the help system easier.
 - Step 3** [Creating the Help Topic Files](#)—The Cisco writer’s guides contain procedures for creating help topics in both authoring environments.
 - Step 4** [Maintaining Your Help System’s Mapping File](#)—All help systems must have a mapping file. If one does not exist, the help writer must create one. If new dialog boxes are created, the mapping file must be updated.
 - Step 5** [Maintaining the Search Index File](#)—Any time the contents of your help system change, you must update the search index file to reflect these changes.
 - Step 6** [Delivering Your Help System](#)—Depending on the authoring environment, some cleanup tasks may be required.
-

CISCO CONFIDENTIAL

Selecting an Authoring Tool

Cisco help was designed to allow for more than one authoring method. You can use any of the following methods to create online help:

- **Native HTML Authoring:** Writers use a native HTML authoring tool, such as HTML Help Workshop, to create topics, contents, and index, save to HTML, then convert to Cisco Online Help format. Writers using a native HTML authoring tool should download the following document from the [Cisco Online Help support website](#): *Creating Online Help Using Native HTML*, EDCS-298882.
- **XML Authoring:** Writers use an XML authoring tool, such as XMetal, to create topics, contents, and index, save to XML, then convert to Cisco Online Help format. Writers using an XML authoring tool should download the following document from the [Cisco Online Help support website](#): *Creating Online Help Using XML*, EDCS-357012.
- **FrameMaker/WebWorks Authoring:** This method permits “single-source” authoring. Writers create text in FrameMaker using the Cisco corporate online help FrameMaker templates. These documents can be printed as manuals for the product and posted to Cisco.com and the documentation CD. Then the writer creates HTML help using Quadralay’s WebWorks Publisher and a custom WebWorks template, and includes the output in the product as online help. Writers using the FrameMaker/WebWorks authoring method should download the following documents from the [Cisco Online Help support website](#):
 - *Creating Cisco Online Help Using FrameMaker and WebWorks*, ENG-104742
 - *Guidelines for Writing Single-Sourced Manuals*, ENG-70452

**Note**

Authoring in RoboHelp is no longer supported. Authors should convert existing RoboHelp projects to native HTML and then use the native HTML authoring process, as explained in [EDCS-298882](#).

Setting Up Your Authoring Environment

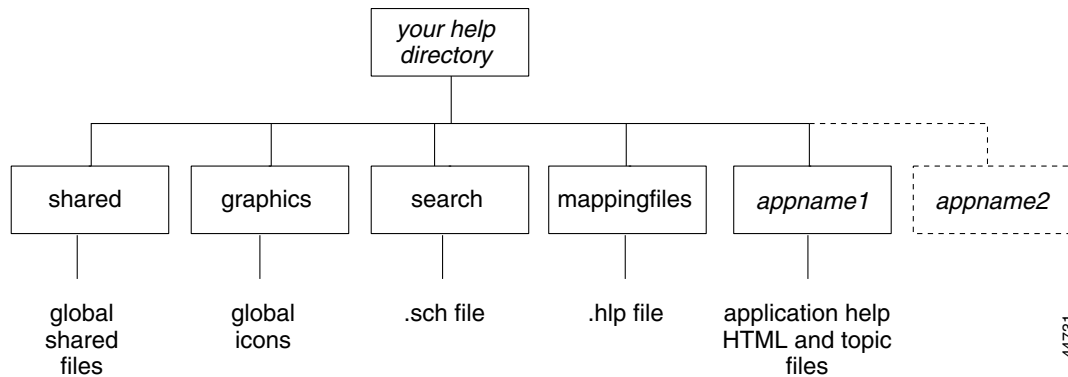
The structure of your local writing environment is determined by the authoring tool you have selected. These topics describe the recommended environment for each authoring tool:

- [Setting Up the Native HTML Authoring Environment](#)
- [Setting Up the XML Authoring Environment](#)
- [Setting Up the FrameMaker/WebWorks Authoring Environment](#)

Setting Up the Native HTML Authoring Environment

Use the instructions in the following document to set up your writing environment: [Creating Online Help Using Native HTML](#), EDCS-298882.

When your environment is set up, (typically, on your local machine), it should look like [Figure 16-7](#).

CISCO CONFIDENTIAL**Figure 16-7 Local Native HTML Writing Environment**

Each of the subdirectories under the main help directory contains specific files:

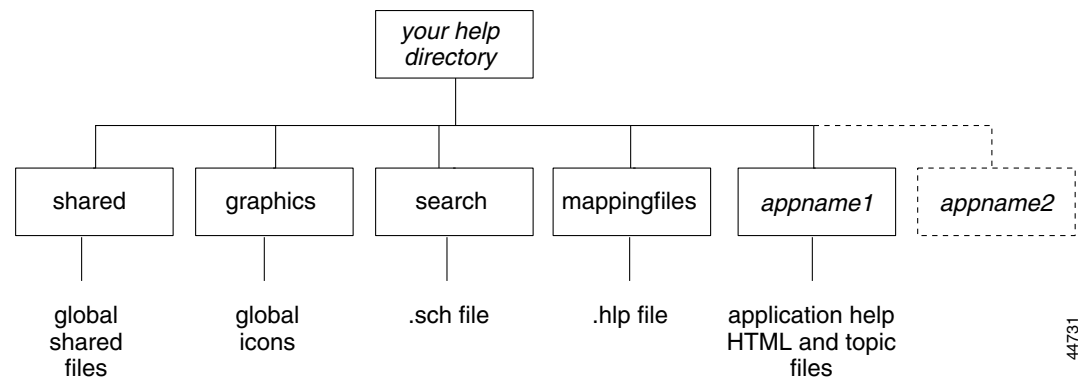
- **shared**—Contains resources shared globally across help systems, such as the login and Using Help topics and style sheets.
- **graphics**—Contains all graphics shared globally across help systems, such as the gifs used in the banner frame.
- **search**—These subdirectories are not required in a typical local writing environment unless you have also downloaded the help or search engine files and are using them to test your help files. They do, however, allow you to keep the mapping and search index files separate from your topic files and reflect the runtime environment:
 - **mappingfiles**—Contains the mapping file for this help package.
 - **search**—Contains the search index file for this help package.
- **appname**—where *appname* is the name you are using for this help system. Contains the help files for this application. Any images that are only referenced by this application are stored in the images subdirectory.

If you are working on help for more than one application and therefore have more than one native HTML project, you can add application subdirectories for each help system and share the other subdirectories (search, shared, graphics, and so on). In this case, your search and mappingfiles subdirectories will contain one .sch and one .hlp file for each help system.

Setting Up the XML Authoring Environment

Use the instructions in the following document to set up your writing environment: [Creating Online Help Using XML, EDCS-357012](#).

When your environment is set up, (typically, on your local machine), it should look like [Figure 16-7](#).

CISCO CONFIDENTIAL**Figure 16-8 Local XML Writing Environment**

Each of the subdirectories under the main help directory contains specific files:

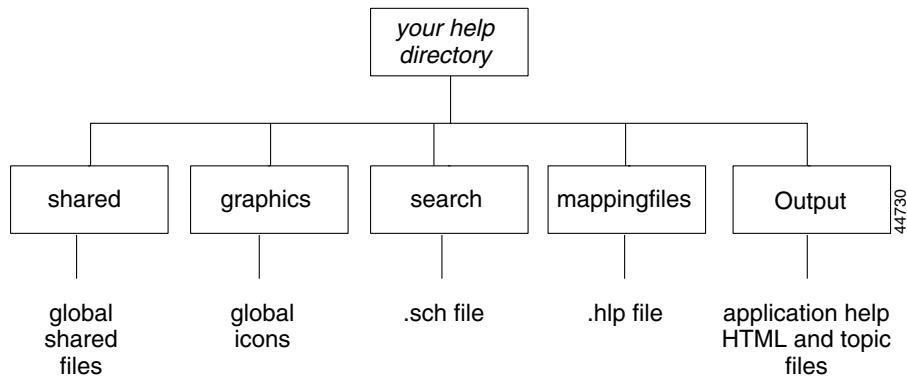
- **shared**—Contains resources shared globally across help systems, such as the login and Using Help topics and style sheets.
- **graphics**—Contains all graphics shared globally across help systems, such as the gifs used in the banner frame.
- **search and mapping files** —These subdirectories are not required in a typical local writing environment unless you have also downloaded the help or search engine files and are using them to test your help files. They do, however, allow you to keep the mapping and search index files separate from your topic files and reflect the runtime environment:
 - **mappingfiles**—Contains the mapping file for this help package.
 - **search**—Contains the search index file for this help package.
- **appname**—where *appname* is the name you are using for this help system. Contains the help files for this application. Any images that are only referenced by this application are stored in the images subdirectory.

If you are working on help for more than one application and therefore have more than one XML project, you can add application subdirectories for each help system and share the other subdirectories (search, shared, graphics, and so on). In this case, your search and mappingfiles subdirectories will contain one .sch and one .hlp file for each help system.

Setting Up the FrameMaker/WebWorks Authoring Environment

Use the instructions in the following document to set up your writing environment: [Creating Cisco Online Help Using FrameMaker and WebWorks, ENG-104742](#).

When your development environment is set up (typically, on your local machine), it should look like [Figure 16-9](#).

CISCO CONFIDENTIAL**Figure 16-9 Local FrameMaker/WebWorks Writing Environment**

Each of the subdirectories under the main help directory contains specific files:

- **shared**—Contains resources shared globally across help systems, such as the login and Using Help topics and style sheets.
- **graphics**—Contains all graphics shared globally across help systems, such as the gifs used in the banner frame.
- **search and mappingfiles**—These subdirectories are not required in a typical local writing environment unless you have also downloaded the help or search engine files and will be using them to test your help files. They do, however, allow you to keep the mapping and search index files separate from your topic files and reflect the runtime environment:
 - **mappingfiles**—Contains the mapping file for this help package.
 - **search**—Contains the search index file for this help package.
- **Output**—Contains the help files for this application. Any images that are only referenced by this application are stored in the images subdirectory.

Store your FrameMaker source files, WebWorks template file (wfp), and illustrations in the top-level help directory. When you run WebWorks, it will automatically copy your help topics and supporting files to the Output subdirectory.

**Caution**

If you are working on help for more than one application, *do not try to share the subdirectories* (search, shared, graphics, and so on). If you try to share subdirectories, WebWorks will overwrite the Output directory each time you run it. Instead, set up a separate directory structure for each help system.

Creating the Help Topic Files

Use these manuals to help you create your help topics:

- Native HTML authors: [Creating Online Help Using Native HTML, EDCS-298882](#).
- XML authors: [Creating Online Help Using XML, EDCS-357012](#).
- FrameMaker/WebWorks authors: [Creating Cisco Online Help Using FrameMaker and WebWorks, ENG-104742](#), and [Guidelines for Writing Single-Sourced Manuals, ENG-70452](#).

In addition, use these guidelines when working in your local authoring environment:

CISCO CONFIDENTIAL

- **Native HTML and XML authors:** Before testing or delivering your files, run the Cisco Online Help Generator to convert your files into Cisco Online Help.
- **FrameMaker/WebWorks authors:** The Output subdirectory (see [Figure 16-9](#)) will contain *all the files* you need for your online help. FrameMaker/WebWorks authors do not need to run the Cisco Online Help Generator.

**Note**

Authoring in RoboHelp is no longer supported. Authors should convert existing RoboHelp projects to native HTML and then use the native HTML authoring process, as explained in [EDCS-298882](#).

Maintaining Your Help System's Mapping File

Mapping files require either inputs from both the engineer and the writer, or an agreement that the writer will create the engineering tags in the mapping file and the engineer will add these tags to the source code.

The following topics describe how to create and maintain your mapping file:

- [Creating the Mapping File](#)
- [Defining the Main Help Page Contents and Index](#)
- [Adding Search Support](#)

Creating the Mapping File

**Note**

If you are using FrameMaker and WebWorks, follow the instructions for inserting markers to create a mapping file in [Creating Cisco Online Help Using FrameMaker and WebWorks, ENG-104742](#). You can download a copy from the [Cisco Online Help support website](#).

If you are using Native HTML or XML authoring, follow this procedure to create a mapping file for your help system:

- Step 1** Create the mapping file manually using a text editor. You can download a template from the [Cisco Online Help support website](#).
- Step 2** Name your mapping file. The file name must be unique and use the .hlp suffix. Follow the naming conventions described in the “[Mapping File Conventions and Requirements](#)” section on page 16-10.
- Step 3** Add a tag/filepath pair for each dialog box that has a Help button. Follow the conventions described in the “[Mapping File Conventions and Requirements](#)” section on page 16-10.
- Step 4** Add a DROPIN or a DROPINPLACE line to define the entry for your help system in the main help contents. For more information about these tags, see the “[Defining the Main Help Page Contents and Index](#)” section on page 16-24.
- Step 5** Add a SEARCH line to define the entry in the search scope list and the search index files to be searched when the user selects that entry. For more information about implementing search, see the “[Adding Search Support](#)” section on page 16-27.
- Step 6** If you are using the NMTG ClearCase processes, run the BOM file creation utility, createbom.exe, to add the mapping file to the BOM.

CISCO CONFIDENTIAL

If you are *not* using the NMTG ClearCase processes, copy the mapping file to the following runtime directory:

`NMSROOT/htdocs/help/mappingfiles`

where `NMSROOT` is the directory in which the product is installed.

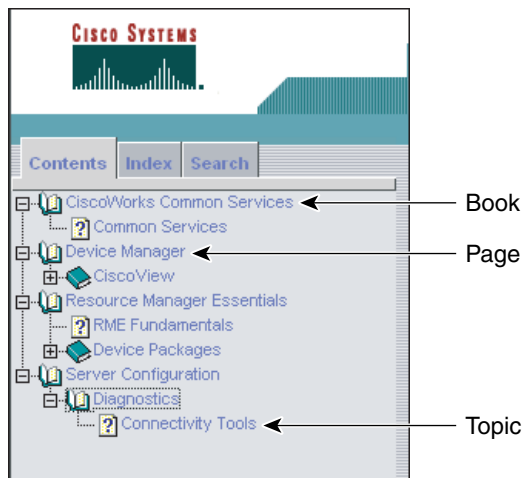
Defining the Main Help Page Contents and Index

The Cisco help engine reads the DROPIN and DROPINPLACE lines in the mapping files to create the Main Help page index and contents.

- The index contains links, in alphabetic order by application, to the default page of each installed help system.
- The Main Help page contents can have multiple levels, represented by expanding and collapsing folders represented by book icons. The Cisco help engine, by default, alphabetizes the contents entries in each level.

Figure 16-10 shows the contents for a sample Main Help page.

Figure 16-10 Main Help Page Contents



115442

Although the order of the index entries is fixed, the writer can determine the order in which the contents entries appear.

The DROPIN and DROPINPLACE lines in the mapping files determine the entries that appear in the main help contents and their relative locations. The following topics discuss these special lines:

- [Adding Contents Entries in the Default Order](#)
- [Adding Contents Entries in a Specific Order](#)

Adding Contents Entries in the Default Order

Use DROPIN lines to add a link in the Main Help page contents to your help system when the *order* in which the help system entry appears in the list of book entries is *not important*.

CISCO CONFIDENTIAL

Note You can put DROPIN lines anywhere in the file, but it is best to put them before any tag/filepath pairs.

The DROPIN line requires this syntax:

```
DROPIN, "level 1 name" [,"level 2 name" ] [...] [ , /filepath]
```

The number of level name entries in the DROPIN line determines the location of the link in the contents hierarchy. The following examples illustrate different ways to use the DROPIN line:

Example 1

```
DROPIN, "Server Configuration", "Desktop", /CWCS/index.html
```

Because this DROPIN line contains two entries, “Server Configuration” and “Desktop,” the Cisco help engine will create a book called “Server Configuration” that contains the page, “Desktop.”

Example 2

```
DROPIN, "Server Configuration", "Administration", "Basic Administration", /admin/index.html
```

This DROPIN line contains three entries: “Server Configuration,” “Administration,” and “Basic Administration.” The Cisco help engine will create a book called “Server Configuration” that contains a second book called “Administration.” The book “Administration” will contain the page, “Basic Administration.”

Example 3

```
DROPIN, "Resource Manager Essentials", "24-Hour Reports", "Software Upgrade Report",  
/swim/sw_hist_24hr.html
```

(all on one line)

You can also create a book in the contents that contains links to topics from many different help systems. This DROPIN line creates a book in the “Resource Manager Essentials” book called “24-Hour Reports” and adds the page, “Software Upgrade Report” to this book. Other help systems that must add an entry to the “24-Hour Reports” book can add a similar line to their mapping files. For example, in its mapping file, the Syslog help system adds the report “Syslog Messages” to the “24-Hour Reports” book:

```
DROPIN, "Resource Manager Essentials", "24-Hour Reports", "Syslog Messages",  
/syslog/sa_uinfo.html
```

(all on one line)



Note Because the file names in these examples are *not* index.html, the report will appear in the right frame instead of replacing the entire window; the top-level contents will remain in the left frame.

Related Topics

- [Understanding Mapping Files](#)
- [Adding Contents Entries in a Specific Order](#)

Adding Contents Entries in a Specific Order

Use the DROPINPLACE keyword to force book or page entries for a help system to appear in a specific location in that book in relation to other entries at the same level.

CISCO CONFIDENTIAL

Note You can put DROPINPLACE lines anywhere in the file, but it is best to put them at the top, before any tag/filepath pairs, for easy access.

The DROPINPLACE line requires this format:

```
DROPINPLACE, "order#", "level 1 name" [, "level 2 name" ] [...][, /filepath]
```

The Cisco help engine applies the *order#* field to all entries *at the same tree level*. The order number default value is 999. Order numbers *lower* than 999 will appear *before* book and page entries that use the default. Order numbers *higher* than 999 will appear *after* book and page entries using the default.

Table 16-6 illustrates this logic.

Table 16-6 Order Number Logic

Order Number	Location
1	First in list
2	Second in list
3...	Third in list ...
999	Default level when no DROPINPLACE tag is used
1000	Third to last in list
1001	Second to last in list
1002...	Last in list ...

Use these guidelines when adding a DROPINPLACE line to your mapping file:

- To force entries to the top of the list, assign the *order#* field values 1, 2, 3...
- To force entries to the bottom of the list, assign values greater than 999.
- The Cisco help engine applies the order number to the *last entry* in the list of level names. In this example, the order number “1” applies to the “Using Campus Manager” entry:

```
DROPINPLACE, "1", "Campus Manager", "Using Campus Manager", /CMcore/UGuide/index.html  
(all on one line)
```

- The Cisco help engine applies the DROPINPLACE order numbers to all entries in a book at the same level.
- To control the order of the top-level book entries, add another DROPINPLACE line to any mapping file. For example, the following line will ensure that the “Server Configuration” book always appears first in the list of top-level books:

```
DROPINPLACE, "1", "Server Configuration"  
(all on one line)
```

- If more than one DROPINPLACE line per tree level contains the same order number, those names will be displayed in alphabetic order.

The order number you use determines the location of your link. The following examples illustrate different ways to use the DROPINPLACE line:

CISCO CONFIDENTIAL

Example 1

User guides should appear at the top of the entries for a product suite. To add the user guide for Campus Manager, add the following line to the Campus Manager user guide mapping file:

```
DROPINPLACE, "1", "Campus Manager", "Using Campus Manager", /CMcore/UGuide/index.html
```

(all on one line)

The number 1 tells the Cisco help engine to place the “Using Campus Manager” entry at the top of the Campus Manager contents entries.

Example 2

To move the 24-Hour Reports book to the bottom of the Resource Manager Essentials list, add the following line to the RMcore mapping file:

```
DROPINPLACE, "1001", "Resource Manager Essentials", "24-Hour Reports"
```

(all on one line)

The number 1001 tells the Cisco help engine to place this entry at the bottom of the list of Resource Manager Essentials entries (books and pages). Because this is a book in the contents, a filepath is not required.

Related Topics

- [Understanding Mapping Files](#)
- [Adding Contents Entries in the Default Order](#)

Adding Search Support

To implement search for a new help system:

Step 1 If you are using the Native HTML or XML authoring tools to create online help, you will need to convert it to a Help project using the Cisco Online Help Generator tool. See the *Creating Cisco Online Help Using Native HTML* or *Creating Cisco Online Help Using XML* documents for details on the conversion.

Step 2 If you are using the WebWorks authoring tool to create online help, the search index file is created as part of the WebWorks project. See “Creating the Initial WDT” section for your help engine in the *Creating Cisco Online Help Using FrameMaker and WebWorks* document if you want to customize your search parameters.

The new search index file has an .sch file extension. The default name is your mapfile name; for example, avmgr.sch.

Step 3 Ensure you move the search index file to the search directory.

Step 4 If you are using NMTG Clearcase processes, ensure the BOM file is where it should be at runtime (default is `install_path/help/search`).

If you are *not* using the NMTG ClearCase processes, copy the search index file to the following runtime directory:

```
NMSROOT/htdocs/help/search
```

where *NMSROOT* is the directory in which the product is installed.

Step 5 Ensure the SEARCH line is in the mapping file.

For example, the search line for the Availability help system would look like this:

```
SEARCH,"Availability", "avmgr.sch"
```

CISCO CONFIDENTIAL

In this example, when the user selects Availability from the search scope list, the search engine looks at the Availability search index file.

Related Topics

- [Understanding Mapping Files](#)
- [Maintaining the Search Index File](#)

Maintaining the Search Index File

When the contents of your help system change, you must update the search index file to reflect these changes. Each authoring tool updates the search index file by re-reading all the help topics for your application and replacing the old file with a new search index file.

To update the search index file:

-
- Step 1** If you are using native HTML or XML, regenerate your help files with your authoring tool. Then use the Cisco Online Help Generator to convert the regenerated files.
 - Step 2** If you are using WebWorks, you need to regenerate your output.
 - Step 3** Ensure you copy the search index file to the search directory to replace the old file.
-

Delivering Your Help System

When you deliver your files—whether that means adding them to ClearCase or copying them directly to the build machine or putting them in a location that the build engineer can access—you should understand the help runtime environment requirements.

When you copy your files, follow these guidelines. Refer to [Figure 16-6](#) for the help runtime directory structure, and [Figure 16-7](#), [Figure 16-9](#) and [Figure 16-8](#) for the local development directory structures.

- Copy *the contents* of your help topic directory (*appname* or Output), including the images subdirectory, to the help *appname* runtime directory.



Note FrameMaker/WebWorks authors—*do not name the runtime directory “Output.”* The runtime directory name must be unique.

- Copy your mapping file (.hlp) from your mappingfiles directory to the corresponding help runtime directory.
- Copy your search index file (.sch) from your search directory to the corresponding runtime directory.
- Unless you are setting up the shared and graphics directories, *do not* copy any files from your local system to these help runtime directories.

CISCO CONFIDENTIAL

Adding Drop-In Help Systems

“Drop-in help” refers to online help for in-house or third-party products that are not part of the CiscoWorks software release, but for which access from the main help window is required. For example, a customer who wants access to HP OpenView online help can add this functionality to the installed help system.

Merging in-house and third-party drop-in help into the main help system requires, at a minimum:

- Supplying a help system that can be viewed in a web browser (HTML-based help)

If possible, rename the first page of the drop-in help system to index.html. If the first page uses this name, the Cisco help engine replaces the main help page with the drop-in help system. If the first page is not called index.html, the drop-in help system appears in the right frame of the help window.

- Supplying a mapping file

Even if the drop-in help system does not supply context-sensitive help or uses a different method, you must supply a mapping file that contains a DROPIN line. This line defines the location of the link to the help system in the main help page contents and index (see the [“Updating the Mapping File”](#) section on page 16-15).



Note The user still has access to the drop-in help system’s existing context-sensitive help even if a link is not added to the main help page.

- Asking the engineer to add the HELPURL tag to the application registry file for the drop-in application. This allows the user to select the application task from the desktop navigation tree and click Help to directly access the help system for that task without browsing the main help page.

The HELPURL tag in the application registry file accepts either a tag or a filepath. For drop-ins, enter the filepath for the help system in this field.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 17

Using the Daemon Manager

The CWCS Daemon Manager provides the following services:

- Maintains the startup dependencies among processes
- Starts and stops processes based on their dependency relationships
- Restarts processes if an abnormal termination is detected
- Monitors the status of processes

The Daemon Manager is useful to applications that have long-running processes that must be monitored and restarted, if necessary. It is also used to start processes in a dependency sequence and to start transient jobs.

The following topics describe the Daemon Manager and how to use it in your applications:

- [Understanding the Daemon Manager](#)—Explains the basic concepts of the Daemon Manager and what it can do.
- [Using the Daemon Manager](#)—Provides guidelines on using the Daemon Manager in CLI, C, C++, and Java environments.
- [Daemon Manager Command Reference](#)—Describes the Daemon Manager CLI, C, C++, and Java methods and commands.

For more information about the Daemon Manager, refer to the engineering specification (EDCS document number ENG 21240).

This specification is also available at the following URL:

http://wwwin-eng.cisco.com/Eng/ENM/BG_10/Specs/BG1PProcessMgrSpec.doc

Understanding the Daemon Manager

Applications use a programmatic interface to the Daemon Manager to register and control processes. The Daemon Manager provides interfaces from C, C++, and Java, and through a set of command line utilities. CWCS also provides a GUI for process monitoring, shutdown, and startup.

A process opens a socket connection with the Daemon Manager to:

- Provide status information to the server
- Receive messages concerning changes to the CWCS Server environment

The Daemon Manager can create and start processes:

- Automatically, at startup time: These are called regular processes (or regular daemons), which means they run all the time.

CISCO CONFIDENTIAL

- As needed (for example, by using inetd): These are called transient processes (or transient daemons).

Applications obtain and save run-time configuration information using a standard set of environment variables. The full set of predefined variables is defined in ENG-21240.

Using the Daemon Manager

You can access the Daemon Manager either from the command line or by using an API written in C, C++, or Java. The following topics describe how to create an application that uses the Daemon Manager:

- [Starting and Stopping the Daemon Manager](#)
- [Using the Daemon Manager Command Line Interface](#)
- [Using the Daemon Manager Application Programming Interface](#)
- [Using the Daemon Manager C++ Interface](#)
- [Using the Daemon Manager Java Interface](#)
- [Using a Ready File to Ensure Process Dependencies are Met](#)
- [Writing Messages to Log Files](#)

Starting and Stopping the Daemon Manager

On a UNIX Platform

-
- Step 1** Log in as root.
- Step 2** To start the Daemon Manager, enter
- ```
/etc/init.d/dmgt d start
```
- Step 3** To stop the Daemon Manager, enter
- ```
/etc/init.d/dmgt d stop
```



Note You cannot start the Daemon Manager if there are non-SSL compliant applications installed with the web server running in SSL-enabled mode.

On a Windows Platform

-
- Step 1** Open a DOS window.
- Step 2** To start the Daemon Manager, enter
- ```
net start CRMdmgt d
```
- Step 3** To stop the Daemon Manager, enter
- ```
net stop CRMdmgt d
```

CISCO CONFIDENTIAL

Using the Daemon Manager Command Line Interface

The Daemon Manager provides several command line interface (CLI) commands. These commands, which are typically used at install time or for debugging purposes, allow you to query the status of a process. They also help you to stop, start, register, or unregister a process.

The CLI commands can be run from a command line interface or using a web browser. They are summarized in the [“Daemon Manager Command Line Utilities”](#) section on page 17-6.

**Note**

The commands to register and unregister a process are only available from the command line interface. They cannot be accessed using a web browser.

Using the Daemon Manager Application Programming Interface

To use the Daemon Manager API:

-
- Step 1** Include the header **dmgt.h** in your C, C++, or Java application.
 - Step 2** After your application has finished its initialization and before it goes into its main loop, instantiate the connection to the Daemon Manager.
 - The C and C++ commands are summarized in the [“Daemon Manager ANSI C and C++ Commands”](#) section on page 17-11.
 - The Java methods are summarized in the [“Daemon Manager Java Methods”](#) section on page 17-17.
 - Step 3** Send a status message (such as the initialization completed successfully or initialization failed and the reason for the failure).
 - Step 4** Update your main loop to process messages from the Daemon Manager. If you are not interested in any messages, redirect all messages to the default message handler routine.
 - Step 5** If the status of your application changes, send a new message to the Daemon Manager so that users can be warned about any problems. This may duplicate some status messages that are already being sent to the system logging service. However, this helps serviceability because the “Last” status message is easily retrieved via the Daemon Manager interface.
-

Use these guidelines when you register your process under the Daemon Manager:

- Programs should not fork or execute themselves.
- Programs should exit when they receive SIGTERM from the Daemon Manager.
- Programs should create core files only in well-known locations. The Daemon Manager starts processes from the directory where the executables reside.

CISCO CONFIDENTIAL

Using the Daemon Manager C++ Interface

Follow these guidelines when you create a C++ application that interfaces with the Daemon Manager:

- Instantiate the C++ singleton object after completing program initialization. The instantiation should exist as long as the program is active.
- Any message loop that is created must listen to the Daemon Manager and deal with any messages. Send any messages not processed by the application to the default handler `dMgr::ProcessMsg()`.
- The program must not fork or execute itself.
- The program must not perform daemon actions, such as trying to restart itself after it fails. Only the Daemon Manager should do this.
- After initializing, run the command, `SendOkMsg()` (see the “[SendOkMsg](#)” section on page 17-22).

Related Topics

- The ANSI C and C++ commands are summarized in the “[Daemon Manager ANSI C and C++ Commands](#)” section on page 17-11.
- For examples that use the C++ interface, refer to these files in the CodeSamples directory on the SDK CD:
 - `daemon1.cpp`
 - `sample1.dsp`
 - `sample1.cpp`

Using the Daemon Manager Java Interface

The `cwjava` command provides a controlled run-time environment for CWCS-based Java server applications. For information about launching a Java application, see:

- [Understanding the Java Application Launch Process, page 4-1](#)
- [Launching a Java Application, page 4-2](#)

Related Topics

- The methods you can use to manage processes in Java applications are summarized in the “[Daemon Manager Java Methods](#)” section on page 17-17.
- For examples that use the Java interface, see these files in the CodeSamples directory on the SDK CD:
 - `daemon1.java`
 - `sample1.java`

Using a Ready File to Ensure Process Dependencies are Met

If you have a process that takes a long time to start up, and has many dependent processes that start up quickly, you may experience problems with Daemon Manager starting the dependent processes before the underlying process has completed initialization.

CISCO CONFIDENTIAL

In cases like these, call `dMgtCreateReadyFile` (for C and C++ processes) or `CreateReadyFile` (for Java) immediately after the underlying process' initialization code segment. Daemon Manager will wait until the API is called and the Ready file created before starting up any dependent processes. For more information on these Daemon Manager APIs, see the “[dMgtCreateReadyFile](#)” section on page 17-12 and the “[CreateReadyFile](#)” section on page 17-18.

If you want to use a Ready file to ensure startup dependencies are met, you must also specify a timeout value for the Ready file creation process when it first registers with Daemon Manager. You can do this using `DmgrRegisterWR` (for C and C++ processes) or `DmgrRegisterJavaWR` (for Java processes) . For more information on these installation APIs, see the “[DmgrRegister](#)” section on page 21-51.

Ensuring that underlying processes are fully initialized is especially important if your application uses multiple database engines on which other processes depend. To ensure that this is possible, the CWCS database APIs provide an alternative database monitor class, `DBPing`. For more information about how to use `DBPing` in combination with the Ready file creation and special “WR” registration APIs, see the “[DBPing](#)” section on page 11-56.

Writing Messages to Log Files

There are two types of log files:

- The Daemon Manager log contains information regarding process start, termination, and Daemon Manager warning and error messages.
- The Application log stores information logged by an application. To write a message to the application log, direct the output to `stderr/stdout`.

The location of the log files is determined by the operating system:

- On Windows platforms:
 - The Daemon Manager log is located under `NMSROOT/log/syslog.log`.
 - Each application has its own application log under `NMSROOT/log`.
 - The application log has same name as the application with the extension “.log”.
- On UNIX platforms:
 - The Daemon Manager log is located at `/var/adm/CSCOpX/log/dmgtd.log`.
 - All applications share the same application log: `/var/adm/CSCOpX/log/daemons.log`.

Daemon Manager Command Reference

The Daemon Manager provides these interfaces:

- [Daemon Manager Command Line Utilities](#)
- [Daemon Manager ANSI C and C++ Commands](#)
- [Daemon Manager Java Methods](#)

CISCO CONFIDENTIAL**Daemon Manager Command Line Utilities**

Use the CLI commands shown in [Table 17-1](#) to query the status of a process and perform other tasks.

Table 17-1 *Daemon Manager CLI Commands*

Syntax	Description
<code>pdexec AppName</code>	Starts a process
<code>pdrapp AppName</code>	Establishes root access for an application
<code>pdreg -r AppName -e PathName -f flags -d OtherAppName -n -t code</code> <code>pdreg -l AppName</code> <code>pdreg -u AppName</code>	Register, list, or unregister a process
<code>pdshow AppName</code>	Monitors all registered processes
<code>pdterm AppName</code>	Shuts down a process

pdexec

`pdexec AppName`

Starts a process.

Before starting a process, the Daemon Manager examines the dependencies of the starting process and if they have not already been started, starts those processes first.

If the process is being restarted *after a shutdown*, any dependent processes registered with the Daemon Manager is not automatically restarted. Dependent processes are automatically restarted only when the Daemon Manager itself is restarted.

Arguments

AppName [string] Application name.

Return Values

OK Starts the process.

Error Prints an error message on stderr and syslog.

Usage Guidelines

Normal process startup may occur in two ways:

- If the process enables autostart (the default), the server invokes the process when the server is started.
- If the operator invokes a command or runs a shell script to start the application.

pdrapp

`pdrapp AppName`

CISCO CONFIDENTIAL

Establishes root access for an application. This command adds the application name to the file **rootapps.conf** so the application can be launched as root.

Arguments

AppName [string] Application name.

Return Values

OK Appends the application name in the file.

Application name is already present in the file Prints “appname already exists” on stderr and syslog.

pdreg

```
pdreg -r AppName -e PathName -f flags -d OtherAppName -n -t code -w timeout
pdreg -l AppName
pdreg -u AppName
```

The `pdreg` command provides three functions:

- Register a process (-r)—A process must be registered before it can be monitored or controlled by the Daemon Manager.
- Unregister a process (-u)—You must unregister a process if you no longer want the process be managed by the Daemon Manager. The process must be registered again to bring it back under the control of the Daemon Manager.
- Display the registration information of a registered process(-l).



Note After a process is registered, all administrative tasks performed on the process must use the Daemon Manager; otherwise the Daemon Manager information will become unreliable, resulting in an *unstable system*. When a process is registered, stop it using the *pdterm* command; do not terminate it directly.

Arguments

-r *AppName* [string] A descriptive name for the application.

-e *PathName* [string] Fully qualified path name to where the program exists.

-f *flags* [integer] Any startup flags required by the application.

CISCO CONFIDENTIAL

- d *OtherAppName*** [string] List of application names that must be started before *AppName*.
- On UNIX platforms, use a comma-separated list. For example:

```
/opt/CSCOpX/bin/pdreg -r Daemon1 -e /opt/CSCOpX/bin/Daemon1.os -d Daemon2, Daemon3 -f -debug^1^-trace^0 -n
```
 - On Windows platforms, group within quotes. For example:

```
Pdreg -r Daemon1 -e D:\Progra~1\CSCOpX\bin\Daemon1.os -d "Daemon2 Daemon3" -f "-debug 1 -trace 0" -n
```
- n** [string] Do *not* automatically start this application when the Daemon Manager is started. Default: Daemon Manager starts the application.
- t *returnCode*** [string] Specifies the normal return code of a transient process. Upon the termination of a process, its return code is checked against the normal *returnCode*. The process is restarted immediately by the Daemon Manager if they do not match. The *returnCode* can be one of the following values:
- 0—Normal return code is zero (default)
 - n—Normal return code is a negative value.
 - p—Normal return code is a positive value.
 - o—Return code is not checked. No restart is required.
- l *AppName*** [string] Displays the registry contents for an application, the registered processes, and their attributes.
- u *AppName*** [string] Unregisters a process.
- If a server that is being stopped and unregistered is needed by other applications (as specified in the dependency list) the effect is to cascade the shutdown of those servers as well. Dependencies on unregistered (unknown) applications are ignored.
- w *timeout*** [integer] Specifies the maximum amount of time (in milliseconds) that Daemon Manager should wait for the process to finish initialization.
- This timeout value can be passed to `pdreg` during installation using a `DmgrRegisterWR` or `DmgrRegisterJavaWR` API call (see the [“Registering and Unregistering CWCS Daemons”](#) section on page 21-51).

Return Values

- OK -r—Process is registered.
 -l—Registry contents (PathName, flags, OtherAppName, etc.) are displayed.
 -u—Process is unregistered.
- Error Prints an error message on stderr and syslog.

Usage Guidelines

- If the program being registered is autostart, start the process now.

CISCO CONFIDENTIAL

- A *dependency list* defines other programs that must be running for this program to be successfully invoked. If the dependency list is valid, other programs are started if they are not already running.
- Specify dependencies on other servers as a comma-separated list with no blanks. For example:


```
pdreg -r MyServer -e /bin/ls -d AppName1,AppName2
```
- Registering an existing name returns an error. To change an entry, first unregister the process.
- Invalid or nonexisting application names can be registered; they are ignored on startup.
- To register more than one command line flag, or one that requires spaces, use the caret (^) instead of spaces. For example:

On UNIX platforms, enter (all on one line):

```
/opt/CSCOpX/bin/pdreg -r Daemon1 -e /opt/CSCOpX/bin/Daemon1.os -d Daemon2,Daemon3 -f
-debug^1^-trace^0 -n
```

On Windows platforms, enter (all on one line):

```
Pdreg -r Daemon1 -e D:\Progra~1\CSCOpX\bin\Daemon1.os -d "Daemon2 Daemon3" -f "-debug
1 -trace 0" -n
```

After registration, the Daemon Manager tries to start the autostart process if the dependency requirement has been met.

- Java programs should use the `cwjava` executable, which is the CWCS Java2 wrapper. For example, to register a JRM process, use the following statement:

- On UNIX platforms, enter (all on one line):

```
pdreg -r jrm -e /opt/CSCOpX/bin/cwjava -d CmfDbMonitor,RmeOrb,EDS -f
com.cisco.nm.cmf.jrm.Server
```

- On Windows platforms, enter (all on one line):

```
pdreg -r jrm -e C:\PROGRA~1\CSCOpX\bin\cwjava.exe -d "CmfDbMonitor RmeOrb EDS" -f
com.cisco.nm.cmf.jrm.Server
```

Where `com.cisco.nm.cmf.jrm` is the `jrm` package path and `Server.class` is the main program.

To check the possible `cwjava` options, enter the following statement:

- On UNIX platforms, enter:

```
/opt/CSCOpX/bin/cwjava -cw /opt/CSCOpX -help
```

- On Windows platforms, enter:

```
C:\PROGRA~1\CSCOpX\bin\cwjava.exe -cw C:\PROGRA~1\CSCOpX -help
```

Where `-cw` option is used to specify your CiscoWorks2000 installation directory.

pdshow

pdshow *AppName*

Generates a list of all registered processes, their current state (“up”, “down,” and so on), and the attributes used while registering a process.

Arguments

AppName [string] Application name.

CISCO CONFIDENTIAL**Return Values**

OK	Lists all registered processes and their current state.
Error	Prints an error message on stderr and syslog.

Usage Guidelines

If no name is supplied, a list of current applications and their status is sent to stdout. The output detects if the environment is Web-based and displays the status in HTML markup or straight text to a TTY display.

pdterm

pdterm *AppName*

Shuts down a process.

**Note**

Any processes registered with the Daemon Manager as dependents of this process are also shut down. However, if the process is then restarted after a shutdown, these dependent processes are *not* automatically restarted. Dependent processes are automatically restarted only when the Daemon Manager itself is restarted.

Arguments

AppName [string] Application name.

Return Values

OK	Shuts down the process.
Error	Prints an error message on stderr and syslog.

Usage Guidelines

- Normal shutdown of a process may occur in two ways:
 - The operator issues a shutdown to the Daemon Manager for a particular server.
 - The server (if transient) sends a request to the Daemon Manager to shut down.
 The Daemon Manager then kills the application.
- Processes that are registered with the Daemon Manager and started manually and are then connected to the Daemon Manager to report that status cannot be stopped with **pdterm**. The pid value is shown as zero.

CISCO CONFIDENTIAL**Daemon Manager ANSI C and C++ Commands**

Use the ANSI C and C++ programming interfaces shown in [Table 17-2](#) to query the status of a process and perform other tasks. The C++ features are defined in the C++ class wrapper.

Table 17-2 *Daemon Manager ANSI C and C++ Commands*

Returns	Syntax and Description
int	dMgtClose (const dMGTHDL dHandle) Closes a connection
int	dMgtCreateReadyFile (char *AppName, char *errmsg) Creates a Ready file after an underlying process (on which other processes depend) has initialized.
const char *const	dMgtErr (const dMGTHDL hdl) Retrieves the error code after Daemon Manager has returned an error (-1).
int	dMgtGetMsg (const dMGTHDL hdl) Reads the Daemon Manager message and places it in the application's buffer.
int	dMgtInit (dMGTHDL *pdHandle, const char *pszAppName) Establishes connection to the Daemon Manager. The Daemon Manager returns a handle (dMGTHDL) to be used by the client application in subsequent requests.
int	dMgtIsShutdown (const dMGTHDL hdl) Checks the incoming request to determine if it is a STOP command.
void	dMgtProcessMsg (const dMGTHDL hdl) Processes Daemon Manager requests.
int	dMgtSendStatus (const dMGTHDL hdl, dMGT_STATE state, const char *pszStatusMsg) Sends a message and changes the status of the application.
char*	GetConFile (char *pszConfile) Gets the Daemon Manager configuration filename.
int	GetDescriptor (const dMGTHDL hdl) Gets the socket descriptor for the I/O port.
int	GetDmgtHostAndPort (char **ppHost, short *psPort, char **ppErrMsg) Gets the name of the Daemon Manager host and port number. This information is defined in the environment variable PX_DMGT_HOST. If PX_DMGT_HOST is not set, then "localhost" (127.0.0.1) is used.
int	ValidatePgmPath (char *pszPgm, char **ppErrMsg) Checks to determine if the <i>pszPgm</i> is a fully qualified path to a program (that is, /path/filename).

dMgtClose

```
int dMgtClose (const dMGTHDL dHandle)
```

Closes the connection to the Daemon Manager.

CISCO CONFIDENTIAL**Input Arguments**

`dHandle` [dMGTHDL] Handle to the client application. dMGTHDL is defined when the application uses `dMgtInit` to connect to the Daemon Manager (see the “[dMgtInit](#)” section on page 17-13).

Return Values

0 OK.

-1 Invalid handle, message error, etc.).

dMgtCreateReadyFile

```
int dMgtCreateReadyFile (char *AppName, char *errmsg)
```

Creates a Ready file after an underlying process (on which other processes depend) has initialized. The Daemon Manager checks the Ready file to ensure that the underlying process has met the dependency requirements before starting the dependent processes.

Input Arguments

AppName [char] Client application name.

Output Arguments

errmsg [char] ASCII string.

Return Values

0 OK.

-1 Error (failed to open file, etc.).

dMgtErr

```
const char *const dMgtErr (const dMGTHDL hdl)
```

Retrieves the error code after Daemon Manager has returned an error (-1).

Input Arguments

`hdl` [dMGTHDL] Handle to the client application that encountered the error condition. dMGTHDL is defined when the application uses `dMgtInit` to connect to the Daemon Manager (see the “[dMgtInit](#)” section on page 17-13).

CISCO CONFIDENTIAL**Return Values**

Error code	Error code.
"no details on error"	No error code is available.

dMgtGetMsg

```
int dMgtGetMsg (const dMGTHDL hdl)
```

Reads the Daemon Manager message and places it in the application's buffer.

Output Arguments

hdl [dMGTHDL] Handle to the client application. dMGTHDL is defined when the application uses dMgtInit to connect to the Daemon Manager (see the "dMgtInit" section on page 17-13).

Return Values

0	OK.
-1	Invalid handle, read error, etc.

dMgtInit

```
int dMgtInit (dMGTHDL *pdHandle, const char *pszAppName)
```

Establishes connection to the Daemon Manager. The Daemon Manager returns a handle (dMGTHDL) to be used by the client application in subsequent requests.

Input Arguments

pszAppName [char] Client application name in ASCII format.

Output Arguments

pdHandle [dMGTHDL] Handle to the client application.

Return Values

0	OK.
-1	Error.

CISCO CONFIDENTIAL**dMgtIsShutdown**

```
int dMgtIsShutdown (const dMGTHDL hdl)
```

Checks the incoming request to determine if it is a STOP command.

Output Arguments

hdl [dMGTHDL] Handle to the client application. dMGTHDL is defined when the application uses dMgtInit to connect to the Daemon Manager (see the “dMgtInit” section on page 17-13).

Return Values

0	Incoming request is NOT a STOP command.
1	Incoming request is a STOP command.
-1	Invalid handle, message error, etc.

dMgtProcessMsg

```
void dMgtProcessMsg (const dMGTHDL hdl)
```

Processes Daemon Manager requests.

Output Arguments

hdl [dMGTHDL] Handle to the client application. dMGTHDL is defined when the application uses dMgtInit to connect to the Daemon Manager (see the “dMgtInit” section on page 17-13).

Return Values

None

dMgtSendStatus

```
int dMgtSendStatus (const dMGTHDL hdl,  
                    dMGT_STATE state,  
                    const char *pszStatusMsg)
```

Sends a message and changes the status of the application.

CISCO CONFIDENTIAL

Input Arguments

`state` [dMGT_STATE] Application status. Allows these values:

- DMGT_READY: Application is OK.
- DMGT_FAILED: Application failed to initialize.
- DMGT_BUSY: Application does not respond to server message.

`pszStatusMsg` [char] Application message in ASCII format. Maximum size is 120 characters.

Output Arguments

`hdl` [dMGTHDL] Handle to the client application. dMGTHDL is defined when the application uses `dMgtInit` to connect to the Daemon Manager (see the “[dMgtInit](#)” section on page 17-13).

Return Values

0 Operation successfully executed.

-1 Invalid handle, invalid state, failed to write to message buffer.

GetConFile

```
char* GetConFile (char *pszConfile)
```

Gets the Daemon Manager configuration filename.

Output Arguments

`pszConfile` [char] Pointer to the Daemon Manager configuration file.

Return Values

`pszConfile` Pointer to the Daemon Manager configuration file.

GetDescriptor

```
int GetDescriptor (const dMGTHDL hdl)
```

Gets the socket descriptor for the I/O port.

Output Arguments

`hdl` [dMGTHDL] Handle to the client application. dMGTHDL is defined when the application uses `dMgtInit` to connect to the Daemon Manager (see the “[dMgtInit](#)” section on page 17-13).

CISCO CONFIDENTIAL**Return Values**

socket descriptor Socket descriptor for I/O port.

-1 Invalid handle, read error, etc.

GetDmgtHostAndPort

```
int GetDmgtHostAndPort (char **ppHost,
                       short *psPort,
                       char **ppErrMsg)
```

Gets the name of the Daemon Manager host and port number. This information is defined in the environment variable PX_DMGTHOST. If PX_DMGTHOST is not set, then “localhost” (127.0.0.1) is used.

**Note**

The caller has to free the memory pointed to by ppHost, using *free()*.

Output Arguments

ppHost [char] Pointer to Daemon Manager host name.

psPort [short] Pointer to Daemon Manager port number.

ppErrMsg [char] ASCII string to indicate any error.

Return Values

0 OK.

-1 Invalid handle, read error, etc.

ValidatePgmPath

```
int ValidatePgmPath (char *pszPgm,
                    char **ppErrMsg)
```

Checks to determine if the string to be validated is a fully qualified path to a program (that is, /path/filename).

Input Arguments

pszPgm [int] String to be validated.

CISCO CONFIDENTIAL**Output Arguments**

`ppErrMs` [char] ASCII string to indicate any error.

Return Values

0 OK.

-1 Invalid handle, read error, etc.

Daemon Manager Java Methods

Use the public Java interfaces shown in [Table 17-3](#) to query the status of a process and perform other tasks.

Table 17-3 Daemon Manager Java Methods

Returns	Syntax and Description
int	CreateReadyFile (<i>String appName</i>) Creates a Ready file after an underlying process (on which other processes depend) has initialized.
int	GetCmdType () Gets the command type exacted from the message.
Socket	GetDescriptor () Gets the Socket object to listen for messages from Daemon Manager.
String	GetErr () Gets the current “last error text” from the Daemon Manager. This routine is usually called when an error condition is returned from the Daemon Manager.
boolean	GetMsg () Retrieves a Daemon Manager message from the IP stack.
boolean	GetServerInfo (<i>String hostname, Integer portinfo</i>) Gets the Daemon Manager server name and port number.
String	GetStatusMsg () Gets the status of the process from Daemon Manager.
boolean	IsShutdownRequest () Checks to determine if the incoming request is a STOP command.
void	ProcessMsg () Processes Daemon Manager requests.
int	ReqStatus (<i>String appName</i>) Asks Daemon Manager to report the status of the process specified by <i>appName</i> .
int	SendBusyMsg (<i>String StatusMsg</i>) Notifies Daemon Manager that it is ready and running. Also tells the Daemon Manager not to send any broadcast messages to this process. The input string <i>StatusMsg</i> appears in the <code>pdshow</code> output.
int	SendErrMsg (<i>String StatusMsg</i>) Notifies Daemon Manager that the process failed to run. The input string <i>StatusMsg</i> is the reason for the failure and appears in the <code>pdshow</code> output.
int	SendOkMsg (<i>String StatusMsg</i>) Notifies Daemon Manager that it is ready and running. The input string <i>StatusMsg</i> appears in the <code>pdshow</code> output.

CISCO CONFIDENTIAL**Table 17-3** *Daemon Manager Java Methods (continued)*

Returns	Syntax and Description
int	StartProcess (<i>String appName</i>) Asks Daemon Manager to start another process specified by <i>appName</i> .
int	StopProcess (<i>String appName</i>) Asks Daemon Manager to stop another process specified by <i>appName</i> .
int	Status () Gets current dMgt object status after the dMgt constructor is called.

CreateReadyFile

```
int CreateReadyFile (String AppName)
```

Creates a Ready file after an underlying process (on which other processes depend) has initialized. The Daemon Manager checks the Ready file to ensure that the underlying process has met the dependency requirements before starting the dependent processes.

Input Arguments

AppName [string] Application name.

Return Values

0 OK
-1 Error (failed to open file, etc.).

GetCmdType

```
int GetCmdType ()
```

Gets the command type exacted from the message.

Arguments

None

Return Values

Returns the command type.

GetDescriptor

```
Socket GetDescriptor ()
```

Gets the Socket object to listen for messages from Daemon Manager.

Arguments

None

CISCO CONFIDENTIAL

Return Values

Success	Returns the Socket object.
Failure	Null.

GetErr

String **GetErr** ()

Gets the current “last error text” from the Daemon Manager. This routine is usually called when an error condition is returned from the Daemon Manager.

Arguments

None

Return Values

Returns the current “last error text” associated with this process.

GetMsg

boolean **GetMsg** ()

Retrieves a Daemon Manager message from the IP stack. This routine waits (blocks) if no data is pending.

Arguments

None

Return Values

True	Message retrieved.
False	Message not retrieved. Call GetErr() to examine the reason for the failure.

GetServerInfo

boolean **GetServerInfo** (String hostname, Integer portinfo)

Gets the Daemon Manager server name and port number.

Input Arguments

portinfo [integer] The port number to be used for the DMGTD connection.

CISCO CONFIDENTIAL**Output Arguments**

hostname [string] The host name where the DMGTD is located.

Return Values

Always returns true.

GetStatusMsg

String **GetStatusMsg** ()

Gets the status of the process from Daemon Manager.

Arguments

None

Return Values

Returns the current status text associated with this process.

IsShutdownRequest

boolean **IsShutdownRequest** ()

Checks to determine if the incoming request is a STOP command.

Arguments

None

Return Values

0 Incoming request *is not* a STOP command.

1 Incoming request *is* a STOP command.

-1 Invalid handle, message error, etc.

ProcessMsg

void **ProcessMsg** ()

Processes Daemon Manager requests.

Arguments

None

Return Values

None

CISCO CONFIDENTIAL**ReqStatus**

```
int ReqStatus (String appName)
```

Asks Daemon Manager to report the status of the process specified by appName.

Input Arguments

appName [String]—Application name.

Return Values

0 Success. Status report sent.
-1 Failure. Status report not sent.

SendBusyMsg

```
int SendBusyMsg (String StatusMsg)
```

Notifies Daemon Manager that it is ready and running. Also tells the Daemon Manager not to send any broadcast messages to this process. The input string *StatusMsg* appears in the pdshow output (see the “pdshow” section on page 17-9).

Input Arguments

StatusMsg [string] Status message, “Program not listening to dmgtld.”

Return Values

0 Success
-1 Failure

SendErrMsg

```
int SendErrMsg (String StatusMsg)
```

Notifies Daemon Manager that the process failed to run. The input string *StatusMsg* is the reason for the failure and appears in the pdshow output (see the “pdshow” section on page 17-9).

Input Arguments

StatusMsg [string] Status message, “Application failed to initialize.”

CISCO CONFIDENTIAL**Return Values**

0	Success
-1	Failure

SendOkMsg

```
int SendOkMsg (String StatusMsg)
```

Notifies Daemon Manager that it is ready and running. The input string *StatusMsg* appears in the pdshow output (see the “pdshow” section on page 17-9).

Input Arguments

StatusMsg [string] Status message, “Program initialized ok.”

Return Values

0	Success
-1	Failure

StartProcess

```
int StartProcess (String appName)
```

Asks Daemon Manager to start another process specified by *appName*.

Input Arguments

appName [string] Application name.

Return Values

0	Success. Start request sent.
-1	Failure. Start request was not sent.

StopProcess

```
int StopProcess (String appName)
```

Asks Daemon Manager to stop another process specified by *appName*.

CISCO CONFIDENTIAL**Input Arguments**

appName [string] Application name.

Return Values

0 Success. Stop request sent.

-1 Failure. Stop request was not sent.

Status

int **status** ()

Gets current dMgt object status after the dMgt constructor is called.

Arguments

None

Return Values

0 Success. Constructor completed.

-1 Failure. Constructor not completed.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 18

Using the Job and Resource Manager

The Job and Resource Manager (JRM) provides a general-purpose interface that allows applications to:

- Schedule jobs—Jobs are general-purpose and application-defined.
- Lock resources by name—Resource locking is done by name and is advisory; that is, JRM is intended to be a *repository* of the currently locked devices; it does not lock a device.



Note JRM locking is meant to aid *cooperating applications* so they can prevent simultaneously updating the same device.

The following topics describe JRM and how to use it in your applications:

- [Understanding JRM Services](#)
- [Understanding the JRM Architecture](#)
- [Enabling JRM](#)
- [Using JRM from a Java Application](#)
- [Using JRM from a Web Browser](#)
- [Customizing the Job Browser Button Behaviors](#)
- [Using JRM from the Command Line](#)
- [JRM Command Reference](#)

For more information about the Job and Resource Manager, see the Job and Resource Management Functional Specification (BG 1.0/Rigel), ENG 21104.

This document reflects the information found in Revision L of the JRM functional specification, Job and Resource Management Functional Specification (BG 1.0/Rigel), ENG 21104. For the most recent updates to the JRM, refer to the current release of this specification.

CISCO CONFIDENTIAL

Understanding JRM Services

Applications that use Job and Resource Management services can schedule an activity, or job, to occur based on several conditions, including:

- **Launch readiness**— Only one instance of a periodic job can be running at any given time. You need to check whether the job been approved and enabled. Also whether there are any dependent jobs still running. All dependent jobs must finish successfully or new jobs cannot start.

Using JRM, you can schedule a job to run pending approval. For example, device image update operations are often scheduled by network administrators, but must be approved by a manager.

In this case, the network administrator can schedule an update, but it will be run by the time it is scheduled to run only if it has been approved by a manager. Anyone on the list of authorized approvers can review the jobs that require approval and either approve or reject them.

- **Scheduling options**—You can schedule jobs to run once or periodically.

Applications often provide users with the means to schedule a task for a given time. SWIM, for example, lets the user specify when to update a device image. In this case, the application runs on a server, downloads the image to the specified device, and reboots the device. Normally software update tasks are run when the traffic on the device is minimal—for example, 2:00 a.m. Sunday morning. Using JRM services, SWIM can let users schedule a job to run at a specific time.

Another application might need to periodically obtain and analyze device configuration information. Using JRM services, the application can schedule a job to run once a day, once an hour, only on Friday at 3:00 p.m., and so on. JRM also provides the functionality to browse the list of scheduled jobs.

- **Tracking job instances**—JRM helps you track each instances of a recurring job separately. Each instance of the execution of the job will have a unique entry in the job browser. Results of all instances are retained and tracked through entries in the job browser. The instances and the results can be individually purged. The purge policy applies to the instances rather than the entire job.
- **Resource locks**—A resource lock secures a device or device subnode, making it inaccessible for a period of time while a job is performed. Resource locks provide a way to serialize access to a device.



Note JRM is intended to be a *repository* of the currently locked devices; it does not lock a device. JRM locking is meant to aid *cooperating applications* so they can prevent simultaneously updating the same device.

- **Event notification**—Job Management uses the Event Distribution System to post job state and resource state events to other applications. Events can include when a job is started or when it ends; when a job has been canceled, approved or rejected; when a resource has been locked or unlocked; and so on.

JRM also provides job and resource lock attributes that allow applications to create their own customized functionality.

The following topics describe how JRM schedules a job and locks resources:

- [Managing JRM Services](#)
- [Scheduling Jobs](#)
- [Locking Resources](#)
- [Locking Resources from Another Application](#)
- [Locking Parts of a Device](#)

CISCO CONFIDENTIAL**Managing JRM Services**

JRM services use the following logic to schedule and run a job:

1. A job is scheduled (for example, upgrade device image or change device configuration).
2. The job is created and scheduled to run, optionally after approval.
3. At the scheduled time, JRM:
 - a. Determines if the conditions for this job have been met (see the [“Understanding JRM Services” section on page 18-2](#))
 - b. Creates a task to run the job
 - c. Locks resources as it needs to work with them. Locking a resource prevents other jobs that also use JRM's locking functionality from simultaneously updating the same device. When a job is done with a device, it unlocks it explicitly.
Automatic lock release and time-based locking prevent a rogue job from locking the device indefinitely.
 - d. Optionally, reports its progress and sets its completion status.

Scheduling Jobs

A job can be scheduled to run if it is enabled and approved (or does not require approval), and its start time is in the future. Jobs can be scheduled to run once or periodically.

When the scheduled time arrives, the Job Manager checks for the following conditions before running the job:

- The job has been approved and enabled.
- The job is not running. Only one instance of a periodic job can be running at any given time.
- Even if a periodic job does not begin because the start conditions were not satisfied, the job for the next start time will be scheduled anyway.

Table 18-1 summarizes the job scheduling options.

Table 18-1 Job Scheduling Options

Schedule Type	Frequency	Description
Run-once	Start time	Time job is to start.
Periodic	Calendar-based	Specifies the day the job is to be run next. The units can be: <ul style="list-style-type: none"> • Days: Run job every n days. • Weeks: Run job on the given day of the week every n weeks. • Month: Run job on the given day of the month every n months. • Month-end: Run job on the last day of the month every n months. • Month-weekday: Run job on the given day of the first/second/third/fourth/last week every n months.
	Time-based	<ul style="list-style-type: none"> • Start time: Start the next job at a fixed time after the start of the previous invocation. • End time: Start the next job at a fixed time after the finish of the previous invocation.

CISCO CONFIDENTIAL

Locking Resources

Resource locks provide a way to ensure exclusive access to a device. A resource lock secures a device or device subnode, making it inaccessible *to other cooperating applications* using JRM for a period of time while a job is performed.

**Note**

JRM serves as a *repository* of the currently locked devices—an application can ask whether a device it is about to update is being used by another application. *JRM does not lock a device*, which means an application can use a device by just ignoring the fact that it failed to lock the device first. Resource locking is meant to be a means to aid *cooperating applications* so that they can prevent a situation where two applications are simultaneously updating the same device.

When JRM receives a request to lock a resource, it checks the name of the resource against existing locks and performs one of these actions:

- If a resource can be locked, it is added to the locks list.
- If a resource is leased (that is, locked for a certain duration), when the lease expires the resource is unlocked.
- If a resource cannot be locked, the return code indicates this state.

Resources are locked for a certain period of time. When a job cannot estimate how long it will need a resource, it can either:

- Periodically renew the lock.
- or
- Lock the resource, specifying infinite time.

Locking a resource for infinite time is *not recommended*. A lock can be “stuck,” but that only happens when

- A job that locked it does not end (when a job ends, JRM automatically releases all its locks).
- A resource was locked by an application that is not a job. When a resource is stuck, the only remedy is for you to force the lock using the JRM browser.

The resource is unlocked when:

- The job explicitly asks JRM to unlock the resource.
- The lock expires.
- The job that locked the resource has ended.

Locking Resources from Another Application

Although resources are typically locked by jobs run by the JRM server, they can be locked by any application. An application that wants to lock a resource must establish a connection with the JRM CORBA object and request a resource lock by providing the resource path and its ID string.

There are two differences between Job Manager and application resource locks:

- IDs used by the jobs are string representations of the job ID numbers. An ID supplied by an application should start with alphabetic character to avoid ID conflicts.

CISCO CONFIDENTIAL

- JRM automatically unlocks all locks owned by a job on job termination. For applications not locked by JRM, the Job Manager cannot sense that the application ended. Therefore, all the resources locked by an application without specifying a lock time (locked forever) must be explicitly unlocked by that application.

Locking Parts of a Device

Applications that use JRM might need to serialize access to certain parts of the device without necessarily locking the whole device. The device associated with the particular resource must also be easily identified.

JRM's resource naming scheme allows resources to form a hierarchy. The top-level nodes of the hierarchy are fully qualified device names (for example, nm7501.cisco.com) and the subnodes correspond to the parts of the device (for example, card0). Each lock is identified by its resource path, starting from the top level (nm7501.cisco.com/card0).

Locking a particular node prevents other applications from locking any nodes below it and all the nodes on the path to it. For example, if there is a lock for nm7501.cisco.com/card0:

- nm7501.cisco.com/card1 can be locked.
- nm7501.cisco.com cannot be locked.
- nm7501.cisco.com/card0/port0 cannot be locked.

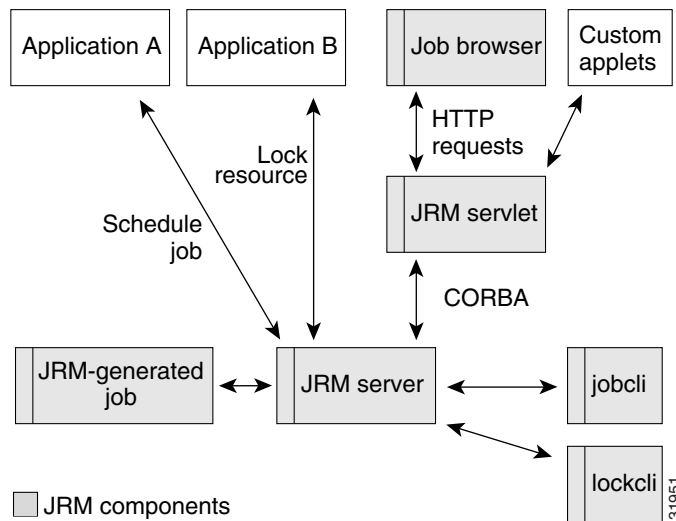
Understanding the JRM Architecture

The following topics describe the JRM architecture:

- [An Overview of the JRM Architecture](#)
- [Understanding the JRM Server](#)
- [Understanding the Job Browser](#)
- [How JRM Relates to Other CWCS Components](#)

An Overview of the JRM Architecture

The following figure shows the relationship between JRM, its components, and its clients.

CISCO CONFIDENTIAL**Figure 18-1 JRM Architecture**

JRM consists of the following components:

- The JRM server provides job and locking services to the following clients:
 - Applications that schedule jobs
 - Jobs that JRM creates in response to a schedule request
 - Applications that lock and unlock resources
 - The JRM servlet
 - The command reference interfaces, jobcli and lockcli, that expose JRM to non-Java applications such as Perl and C++

To learn more about the JRM server, see the [“Understanding the JRM Server”](#) section on page 18-7.

- The JRM servlet—Provides a URL interface to the JRM server process. Since there is no easy way to use CORBA calls directly from a web browser, this intermediary piece of code runs on the server and communicates with the JRM server via CORBA to execute the commands it receives from the web browser. All responses from the JRM servlet are XML-encoded.

The JRM servlet:

- Accepts HTTP requests from the Job Browser and other customized applets and translates them into CORBA calls to the JRM server.
- Accepts JRM server responses, translates them into HTTP responses, and sends them back to the Job Browser or applet.

For information about using the JRM servlet, see the [“Using JRM from a Web Browser”](#) section on page 18-21.

- The Job Browser—A configurable applet that displays the current jobs and locked resources and allows the users to stop, terminate, and remove jobs. The Job Browser applet runs on the web browser and communicates (exchanges XML documents) with the JRM server via the JrmServlet. The Job Browser can be embedded into HTML pages to provide a GUI for browsing and managing JRM jobs and resources.

To learn more about the Job Browser, see the [“Understanding the Job Browser”](#) section on page 18-10.

CISCO CONFIDENTIAL

- Applets and Java components—Used to design custom HTML pages. Since there is a wide variation in GUI requirements for creating and editing different job types, individual applications must provide screens for creating, editing, and displaying details of their job types. To assist in these efforts, JRM provides applets to prompt for the date or time, display the JRM job panel, and display various types of JRM schedules.

For information about designing custom HTML pages, see the [“Customizing the Job Browser Button Behaviors”](#) section on page 18-22.

- jobcli and lockcli applications—Provide a command-line interface for scheduling jobs and locking resources. These applications, which are primarily used for debugging purposes, also provide a JRM interface for non-Java applications such as Perl or C++.

For information about using command line applications, see the [“Using JRM from the Command Line”](#) section on page 18-24.

Understanding the JRM Server

The JRM server provides job and locking services to various clients, including applications that schedule jobs or lock and unlock resources, jobs that JRM creates in response to a schedule request, the JRM servlet, and the command reference interfaces.

The following topics describe the JRM server components:

- [About Jobs and Resources](#)
- [About JRM Server Classes](#)
- [About the IDL Interface](#)
- [About the Helper API](#)
- [About JRM Events](#)

About Jobs and Resources

The two fundamental JRM entities are jobs and resource locks:

- Using JRM, you can schedule a job to run, pending approval. Job information consists of:
 - The command to be run
 - The schedule (run immediately, once, or periodically)
 - An approval flag (approval is required or not required before the job can run)
 - A status string that can be set by the job

Job information is stored in a database table, where each row represents a single job. On startup, JRM reads the table and creates job objects. While running, JRM automatically updates the corresponding row to reflect every change in a job object. When a new job is created, a new row is added to the table. When a job is deleted, the corresponding row in the table is deleted.

- JRM locks resources as it needs to work with them. Resource locks provide a way to ensure exclusive access to a device. Resource lock information consists of:
 - The resource path
 - The owner (a job or any process)
 - The expiration time

CISCO CONFIDENTIAL

Note Resource information is not kept in a database. If JRM is stopped, all resource information is lost.

About JRM Server Classes

The JRM server application class performs these functions:

- Reads configuration files, such as the XML column and action configuration files for the Job Browser.
- Starts various threads. For example, when a job is about to run, it is added to the launch queue. The LaunchQueue thread dequeues the job and invokes the Daemon Manager to run the job.
- Keeps pointers to these JRM implementation classes:
 - JobManagerImpl—A facade to the Jobs class, a singleton containing everything related to jobs. It contains the following inner classes:


```
Jobs.JobTable
Jobs.LaunchQueue
Jobs.Terminator
```
 - LockManagerImpl—A facade to the Locks class, a singleton containing everything related to locks. It contains the inner class, Locks.Lock.
 - AlarmQueue—Maintains the timer queue, runs the timer thread, and invokes a handler when the node timer expires.
 - DBConnection—Provides an interface to database-related functions.
 - DMConnection—Runs a thread that listens to Daemon Manager events and provides a listener interface.
 - EDSConnection—Runs a thread that sends events to EDS.
 - Client—A simplified JRM communication interface for jobs running under JRM (see the “About the Helper API” section on page 18-9).
- Provides utility functions.

About the IDL Interface

The JRM server implements two objects: JobManager and LockManager. Enumeration interfaces are implemented by the iterator objects, JobIter and LockIter. The JRM IDL (Interface Definition Language) file includes the interfaces of the objects supplied by the JRM server.

Related Topics

- For an example of the JRM IDL file, refer to the jrm.idl file in the CodeSamples directory on the CWCS SDK CD.
- For more information about CORBA, refer to:
 - Object Management Group website, <http://www.omg.org>
 - OMG’s CORBA website, <http://www.corba.org>

CISCO CONFIDENTIAL

About the Helper API

The Client class provides a collection of static methods that might be helpful for clients that manipulate jobs and resources. The methods can be categorized by the main class and several inner classes:

- The top-level class, Client, implements Constants. Use this class to return printable representations of the schedule string or the job's run and schedule states. These methods can be used by any client.

The Client class contains these methods, which can be used by any client:

- Return a printable representation of the schedule string
- Return a printable representation of the job's run and schedule states
- Initialize the ORB and locate servers
- The inner class, MyJob, is a collection of static methods that can be used only by a job running under JRM control. This class provides methods to:
 - Set a job's completion state (success, success with info, failed)
 - Set a job's progress string
 - Lock resources
 - Unlock resources
 - Unlock all a job's resources
 - Get job information

Related Topics

See the [“About the Helper API Methods” section on page 18-49](#).

About JRM Events

JRM can use Event Services Software (ESS) and the Event Distribution System (EDS) to publish events of interest to applications. JRM sends events when:

- A job starts.
- A job ends.
- A job start fails.
- A job is canceled.
- A job is rejected or approved.
- A resource is locked or unlocked.
- A process has ended.

These events belong to the event category status (EventCategory_Status). The event and resource atoms are listed in com.cisco.nm.cmf.jrm.JrmEdsAtomDev.java.

JRM publishes the following topics using ESS:

- cisco.mgmt.cw.cmf.jrm.EventJobReject
- cisco.mgmt.cw.cmf.jrm.EventJobEnd
- cisco.mgmt.cw.cmf.jrm.EventJobStart
- cisco.mgmt.cw.cmf.jrm.EventJobCancel
- cisco.mgmt.cw.cmf.jrm.EventLock

CISCO CONFIDENTIAL

- cisco.mgmt.cw.cmf.jrm.EventJobApprove
- cisco.mgmt.cw.cmf.jrm.EventDaemonEnd
- cisco.mgmt.cw.cmf.jrm.EventJobLaunchFail
- cisco.mgmt.cw.cmf.jrm.EventUnlock

JRM publishes the following topics using EDS:

- EventJobStart
- EventJobEnd
- EventJobCancel
- EventJobApprove
- EventJobReject
- EventDaemonEnd
- EventLock
- EventUnlock

JRelated Topics

See:

- [Chapter 19, “Using Event Services Software.”](#)
- [Chapter 20, “Using the Event Distribution System.”](#) Note that EDS is deprecated.

Understanding the Job Browser

The Job Browser is a configurable Java applet that you can embed in HTML pages to provide a GUI for browsing and managing JRM jobs and resources. The Job Browser uses XML files to specify:

- Job and resource table column names, sizes, and visibility.
- URLs to call that carry out the actions entered by the user.

[Figure 18-2](#) shows the Job Browser interface, which provides the user actions shown in [Table 18-2](#).

Figure 18-2 Sample Job Browser Dialog Box

Showing 4 records							
	Job ID	Type	Run Status	Sched Type	Description	Run Sched	Status
1.	1004.1	TJob	Succeeded: 04/23/2004 11:33:16-04/23/2004 11:33:46	Periodic	Test job	Every 1 minutes(s), starting 04/23/2004 11:33:16	FARO: ur too good
2.	1004.2	TJob	Running: 04/23/2004 11:34:16-	Periodic	Test job	Every 1 minutes(s), starting 04/23/2004 11:34:16	
3.	1005	Faro		Scheduled	Once	At 04/23/2004 11:34:45	
4.	1006	Faro		Periodic	Daily	At 11:36:03 daily, starting 04/23/2004	

↑ Select an item(s) then take an action →

Stop Delete Refresh

113542

CISCO CONFIDENTIAL**Table 18-2 Job Browser User Actions**

Action	Description
Stop	Calls the corresponding JRMserver method. Select a Job ID, then click Stop.
Delete	Calls the corresponding JRMserver method. Select a Job ID, then click Delete.
Click a Job ID	Uses the URL registered in the JrmButtonActions.xml file for the selected job type to show job details for the application that owns the job.

Figure 18-3 Sample Job Resource Dialog Box

Showing 3 records				
	Resource ▲	Job Id/Owner	Time Locked	Expire Time
1.	<input type="checkbox"/> res1	faro	Fri Apr 23 11:36:13 IST 2004	Never
2.	<input type="checkbox"/> res2	faro	Fri Apr 23 11:36:13 IST 2004	Fri Apr 23 11:38:13 IST 2004
3.	<input type="checkbox"/> res3	faro	Fri Apr 23 11:36:13 IST 2004	Fri Apr 23 11:41:13 IST 2004

↑--Select an item(s) then take an action-->

Free Resources Refresh

Table 18-3 Job Browser User Actions

Button	Default Action
Free Resource...	<ul style="list-style-type: none"> Explicitly frees resources without waiting for the associated job to end. Frees orphaned resources that no longer have an associated job that is running. <p>Shown only when user has administrative privileges.</p> <p>Note Using this button is <i>not</i> recommended unless resource is orphaned.</p>

Related Topics

- [Customizing the Job Browser Button Behaviors, page 18-22](#)
- [About the Helper API Methods, page 18-49](#)

How JRM Relates to Other CWCS Components

JRM relies on:

- The built-in CWCS database to maintain job states. JRM lists a database as a dependency. Therefore, the Daemon Manager starts JRM only after the database is running. For more on the CWCS database, see [Chapter 11, “Using the Database APIs.”](#)
- The built-in CWCS Daemon Manager to run and control jobs. JRM jobs run as processes under the CWCS Daemon Manager. For more on the CWCS Daemon Manager, see [Chapter 17, “Using the Daemon Manager.”](#)

CISCO CONFIDENTIAL

Enabling JRM

JRM is part of CWCS System Services. Since CWCS release 3.0, JRM services are enabled by default. If your application requires services from JRM, remember to register for this service at installation. For instructions, refer to the “[Registering for CWCS Services](#)” section on page 5-4. If you prefer to request services after installation, refer to the “[Enabling New Service Bundles from the Command Line](#)” section on page 5-5.

Using JRM from a Java Application

To use JRM from a Java application, you must, for example, know how to establish a connection with the Job Manager, create a job, and set the status of the job. The following topics describe some typical job and lock management tasks:

- [Establishing a Connection](#)
- [Creating a Job](#)
- [Setting the Job Status](#)
- [Getting Job Descriptions](#)
- [Handling an Unapproved Job](#)
- [Enabling a Disabled Job](#)
- [Handling a Crashed Job](#)
- [Locking and Unlocking a Device](#)
- [Handling an Unavailable Resource](#)
- [Accessing a Locked Device](#)

For a description of the JRM APIs, see the “[JRM Command Reference](#)” section on page 18-26.

Establishing a Connection

[Example 18-1](#) shows how to establish a connection with the Job Manager. The host where the JRM server is running is passed as a parameter.

**Note**

This example disables automatic rebinding. If automatic rebinding is enabled and the JRM server aborts for any reason, the ORB will try to find another JRM server and reconnect to it. This is not a desirable action.

Example 18-1 Connecting with Job Manager

```
import java.lang.*;
import java.net.*;
import com.cisco.nm.cmf.jrm.*;
import com.cisco.nm.cmf.util.CmfException;
import com.cisco.nm.cmf.util.Util;
```

CISCO CONFIDENTIAL

```

import org.omg.CORBA.*;
import com.inprise.vbroker.CORBA.BindOptions;
import java.util.*;
public class testJpp {
public static void main (String[] args) {
    JrmServiceManager jrm=null;
    JobManager jm=null;
    String nmsroot;
    String host;

    try {
        Util.loadBGProperties("md.properties");
        nmsroot=System.getProperty("NMSROOT");
        System.out.println("NMSROOT is "+nmsroot);
    } catch(CmfException cmf) {
        System.out.println("unable to load md.properties");
    }

    try {

        host=(InetAddress.getLocalHost()).getHostName();
        System.out.println("host = " + host);

        Properties ORBProperties = Client.getOrbConnectionProperties();
        ORBProperties.put("org.omg.CORBA.ORBClass", "com.inprise.vbroker.orb.ORB");
        org.omg.CORBA.ORB orb =
(com.inprise.vbroker.CORBA.ORB)com.cisco.nm.util.OrbUtils.initORB(null,ORBProperties);
        jrm = JrmServiceManagerHelper.bind(orb,Client.getJrmName(),host,null);

        System.out.println("Connected to JRM service Manager.");
        LoginInfo loginInfo = new LoginInfo("admin","admin","");
        jm = jrm.getJobHandle(loginInfo);
    } catch (org.omg.CORBA.SystemException e){
        e.printStackTrace();
        System.err.println(e.toString());
        return;
    } catch (Exception e) {
        e.printStackTrace();
        System.err.println(e.toString());
        return ;
    }

    if (jm == null) {
        System.out.println("Job Manager not bound");
        return;
    }

    //Foll code to create a job

    long start=System.currentTimeMillis()+20000;
    int type=Constants.SCHTYPE_S_Minutes;
    int increment=3;
    Schedule sch=new Schedule(start,type,increment);

    int precedents[]={};

    JobInfo ji =new JobInfo(0,// id
        "TestJob",// type
        new String("Test job"),// description
        "D:\\progra~1\\mkstoo~1\\mksnt\\sleep.exe 30",
        sch,// schedule
        precedents,// dependencies
        Constants.RUNST_NeverRan,// state
        Constants.SCHST_Enabled,// enabled
    );
}
}

```

CISCO CONFIDENTIAL

```

        System.currentTimeMillis(), // Time created
        System.currentTimeMillis(), // Time modified
        0, // Start
        0, // Stop
        "Scheduling Job", // Progress
        host, // Host default=localhost)
        new String("system"), // Account (default=system)
        new String("Reference"), // Reference
        "admin", // Owner
        "" // Approver
    );

    IntHolder jid=new IntHolder(0);

    if(ji == null) {
        System.out.println("Job info is null");
        System.exit(1);
    }

    int status=jm.job_create_hist(ji,jid);

    if(status != Constants.STATUS_Ok) {
        System.out.println("Failed to create Job");
        System.exit(-1); // job creation failed
    } else {
        System.out.println("Job "+jid.value+" created sfly.");
    }
}

```

Creating a Job

Example 18-2 shows how to create a job with these attributes:

- It will run the Java application whose main class is myJavaClass and use a standard Java classpath (that is, the same one that was used to run the JRM server) to locate the classes.
- The job's ID will be passed as a command-line option. A job uses this ID to communicate its progress and completion status to JRM.
- The job's type is ACLM.
- The job requires approval before it can be run.
- The job will run in one minute.

Example 18-2 Creating a Job

```

// Create a job
JRM.Schedule sch =
    new Schedule(System.currentTimeMillis()+60*1000, // Start in a minute
        SCTYPE_Once,
        SCHINC_Months, // Doesn't matter
        0);
    int precedents[] = {};
    JRM.JobInfo ti =
        new JobInfo(          0, // id
            "ACLM", // type
            "Description", // description

```


CISCO CONFIDENTIAL

```

"$JP -cp $JC $JJ myJavaClass",// command:
// run myJavaClass
sch,// schedule
precedents,// dependencies
RUNST_NeverRan,// state
SCHST_RequiresApproval // Approval state:
| SCHST_AM_WAITING // requires approval,
| SCHST_ENABLED,// enabled
    0,// Time created
    0,// Time modified
    0,// Start
    0,// Stop
    "",// Progress
    "",// Host default=localhost)
    "",// Account (default=system)
    "",// Reference
    "",// Owner
    ""// Approver
);

// Create holder for the returned value
IntHolder h_id = new IntHolder(0);

// Create a job, test its status
try {
    int stat = job_manager.job_create(ti,h_id);
    if (STATUS_Ok == stat) {
        System.out.println("Created job with id = ", h_id.value);
    }
    else {
        ...
    }
    catch (org.omg.CORBA.SystemException e) {
        // Attempt to reconnect explicitly
    }
}

```

Setting the Job Status

The following code fragment tells JRM that the job has ended successfully and sets its progress string (which will become the completion string) to “Download successful”:

```

import com.cisco.nm.cmf.jrm.Client;
...
Client.MyJob.set_completion_state(Client.RUNST_Succeeded);
Client.MyJob.set_progress("Download successful");

```

You need to keep the following in mind:

- Execute this code from a job executing under JRM.
- Add \$JJ to the command line that starts this job (see the [“About the Job and Resource Lock Attributes”](#) section on page 18-26).

The displayed job status is a dynamic attribute of the job. JRM calculates the status based on the job’s run state, scheduled state attributes, and the current time.

- For run-once jobs, the displayed job status reflects either:
 - The job’s scheduling state (if the job’s scheduled time is in the future)
 - The job’s run result (if the job’s scheduled time is in the past).

CISCO CONFIDENTIAL

- For periodic jobs, the displayed job status displays the result of the last run and the scheduling state of the next run.

The job status values for both run-once and periodic jobs are summarized in the tables in the [“About Displayed Job Status Values”](#) section on page 18-28.

Getting Job Descriptions

[Example 18-3](#) shows how to get the job descriptions for all scheduled jobs.

Example 18-3 Getting Job Descriptions

```

jrm.JobIterHolder iter = new JobIterHolder();
jrm.JobInfoHolder job_info = new JobInfoHolder();

try
{
    //Get the job enumerator
    int status = job_manager.job_enum(iter);
    if (STATUS_Ok == status)
    {
        while (STATUS_EOF != iter.value.next(job_info))
        {
            System.out.println(job_info.value.szDescription);
        }
        iter.value.release();
    }
}
catch (org.omg.CORBA.SystemException e)
{
    // .....
}

```

Handling an Unapproved Job

Use the code fragment in [Example 18-4](#) when a job that requires approval is scheduled and has not been approved by the scheduled time. The job execution is abandoned, and the job deleted if it is not periodic.

Example 18-4 Handling Unapproved Jobs

```

jrm.JobInfoHolder job_info = new JobInfoHolder();
// Find out the job details corresponding to the job.
int status = job_manager.job_get_info(idJob,job_info);

// If the job is still waiting for approval, then execution of the
// job is abandoned.
if ((STATUS_Ok == status)&&(SCHST_AM_Waiting == (SCHST_AM_Masks &
job_info.value.sch_state)))
{
    System.out.println ("Still waiting for approval, so can't start now");
    jrm.ScheduleHolder schedule = new ScheduleHolder();
}

```

CISCO CONFIDENTIAL

```

// Get the job schedule
int stat = job_manager.job_get_schedule(idJob,schedule);

// Check the schedule type.If the job is not periodic, delete the job
if (STATUS_Ok == stat)
{
if ((SCHTYPE_Immediate == schedule.value.type)|| (SCHTYPE_Once == schedule.value.type))
{
System.out.println ("Now deleting the job") ;
job_manager.job_delete(idJob);
}
}
}

```

Enabling a Disabled Job

[Example 18-5](#) shows how to create a job in the disabled state, do some operations, and then enable the job.

Example 18-5 Enabling a Disabled Job

```

// Create a job
jrm.Schedule sch = new Schedule(0,
    SCHTYPE_Immediate,
    SCHTYPE_Monthly //Ignored for SCHTYPE_Immediate
);

int precedents[] = {};

// Create the JobInfo structure with appropriate values
jrm.JobInfo job_info = new JobInfo(0, // id
    "ACLM", // type
    "Description", // description
    "$JP -cp $JC $JJ myJavaClass", // command:
    // run myJavaClass
    sch, // schedule
    precedents, // dependencies
    RUNST_NeverRan, // state
    SCHST_AM_Approved, // Approval state:
    0, // Time created
    0, // Time modified
    0, // Start
    0, // Stop
    "", // Progress
    "", // Host default=localhost)
    "", // Account (default=system)
    "", // Reference
    "", // Owner
    "" // Approver
);

// IntHolder for holding the JobId
org.omg.CORBA.IntHolder h_id = new org.omg.CORBA.IntHolder(0);
// Create a job, test its status
try
{
    int stat = job_manager.job_create(job_info,h_id);

```

CISCO CONFIDENTIAL

```

if (STATUS_Ok == stat)
{
    System.out.println("Job created with id = "+ h_id.value);
}
else
{
    System.out.println("Job creation failed ");
    System.exit(0);
}
//Perform some operations involving the newly created job

// .....

// Now enable the job
int status = job_manager.job_set_resume(h_id.value,true);
if (STATUS_Ok != status)
{
    System.out.println("Job resumption failed");
}
}
catch (org.omg.CORBA.SystemException e)
{
    System.out.println("Exception while job creation ");
    System.exit(0);
}

```

Handling a Crashed Job

[Example 18-6](#) shows how to get a job's current running state and delete a crashed job.

Example 18-6 Handling a Crashed Job

```

org.omg.CORBA.IntHolder result = new org.omg.CORBA.IntHolder();

int status = job_manager.job_get_result(idJob, result);
if (STATUS_Ok == status)
{
    if (result.value == RUNST_Crashed)
    {
        // Delete the job
        status = job_manager.job_delete(idJob);
        if (STATUS_Ok != status)
        {
            System.out.println("No such job exists");
        }
    }
}
else
{
    System.out.println(" Getting the run state failed!");
}

```

CISCO CONFIDENTIAL

Locking and Unlocking a Device

In [Example 18-7](#), a job locks a device, does some processing, and releases the lock.

Example 18-7 Locking and Unlocking a Device

```

LockManagerImpl lock_manager = new LockManagerImpl("TEST");

int status = lock_manager.lock("device1", "my_app", 1000);

/* If no job has locked device1 yet, then status = STATUS_Ok */
if (STATUS_Ok == status)
{
    System.out.println("No lock exists now for the device ");
    //... do some processing...
    lock_manager.unlock("device1", "my_app");
}

```

Handling an Unavailable Resource

In [Example 18-8](#), a job is enabled and approved and then, at the scheduled time, it tries to lock a resource and fails.

Example 18-8 Handling an Unavailable Resource

```

int status = 0;
jrm.JobInfoHolder job_info = new JobInfoHolder();

try
{
    // Find out the job details corresponding to the job.
    status = job_manager.job_get_info(idJob, job_info);
}
catch (org.omg.CORBA.SystemException e)
{
    System.exit(0);
}

//If the job is approved and is enabled then try to run the job
if ((STATUS_Ok == status)&& (SCHST_AM_Approved == (SCHST_AM_Masks &
job_info.value.sch_state))&&
(SCHST_Enabled == (job_info.value.sch_state & SCHST_Enabled)))
{
    try
    {
        //Lock the required devices
        status = lock_manager.lock("device name", "owner", 1000);

        // If locking failed
        if (STATUS_Ok != status)
        {
            jrm.ScheduleHolder schedule = new ScheduleHolder();
            // Get the job schedule
            int stat = job_manager.job_get_schedule(idJob, schedule);

```

CISCO CONFIDENTIAL

```

    if (STATUS_Ok == stat)
    {
        // If the job is not periodic, then delete the job
        if ((SCHTYPE_Once == schedule.value.type) ||
            (SCHTYPE_Immediate == schedule.value.type))
        {
            System.out.println ("Now deleting the job");
            job_manager.job_delete(idJob);
        }
    }
}
// Run the job
else
{
    status = job_manager.job_run(idJob);
    if (STATUS_Ok != status)
    {
        System.out.println ("job run failed");
    }
}
}
catch(org.omg.CORBA.SystemException e)
{
    System.exit(0);
}
}

```

Accessing a Locked Device

In [Example 18-9](#), a job is trying to lock a device that is already locked by another job. The code finds the information about the other job and, if that job is not running, releases all resources locked by it. Then it tries to lock the device. After the device is locked, the job does some processing and then releases the lock.

Example 18-9 Accessing a Locked Device

```

/* Current job is trying to lock a device device1 */
int status = lock_manager.lock("device1", "my_app", 2000);

/* If some job has already locked device1, then status = STATUS_Exists */
if (STATUS_Exists == status)
{
    /* Find out the complete Lock_info for device1 */
    jrm.LockInfoHolder lock_info = new LockInfoHolder();
    status = lock_manager.get_lock("device1", lock_info);
    /* If Lock_info found for device1, status = STATUS_Ok */
    if (status==STATUS_Ok)
    {
        Integer int_id = new Integer(lock_info.value.szJob);
        int job_id = int_id.intValue();
        jrm.JobInfoHolder job_info = new JobInfoHolder();
        // Find out the job details corresponding to the job id obtained.
        status = job_manager.job_get_info(job_id, job_info);
        if (status==STATUS_Ok)
        {
            if (job_info.value.run_state != RUNST_Running)
            {

```

CISCO CONFIDENTIAL

```

// Release all resources locked by the job
lock_manager.unlock_job(lock_info.value.szJob);
status = lock_manager.lock("device1", "my_app", 1000);
// ... do some processing...
lock_manager.unlock("device1", "my_app");
}
else
System.out.println("No job exists");
}

```

Using JRM from a Web Browser

The JRM servlet provides the URL interface to the JRM server process. The servlet communicates with the JRM server via CORBA to execute the commands it receives. All responses from the JRM servlet are XML-encoded.

Table 18-4 summarizes the URL commands which the JRM servlet supports via HTTP POST and GET requests.

Table 18-4 JRM Servlet URL Commands

URL Command	Description
getJobAndResourceList	Returns an XML-encoded list of currently scheduled jobs and locked resources. Example: <code>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?cmd=getJobAndResourceList</code>
stop	Requests that the specified job be stopped. Returns “true” if successful, “false: <i>error message</i> ” otherwise, where <i>error message</i> provides the message to display to the user. For job history jobs, “instance id” and a boolean variable “stop instance” is used. The “stop instance” should be true if the user selects “Stop this instance only” and false if the user selects “Stop all instances”. Example: For jobs that maintain job history: <code>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?button=stop&jobid=1001&instanceid=2&stopinstance=true</code> For jobs that do not maintain job history: <code>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?button=stop&jobid=1001&instanceid=2&stopinstance=true</code>

CISCO CONFIDENTIAL**Table 18-4** JRM Servlet URL Commands (continued)

URL Command	Description
kill	<p>Requests that the specified job be killed. Returns “true” if successful, “false: <i>error message</i>” otherwise where <i>error message</i> provides the message to display to the user.</p> <p>For job history jobs, “instance id” and a boolean variable “stop instance” is used. The “stop instance” should be true if the user selects “Stop this instance only” and false if the user selects “Stop all instances”.</p> <p>Example:</p> <p>jobs that maintain job history:</p> <pre>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?button=kill&jobid=1001&instanceid=2&stopinstance=true</pre> <p>jobs that do not maintain job history:</p> <pre>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?button=kill&jobid=1001&instanceid=2&stopinstance=true</pre>
remove	<p>Removes specified job from JRM scheduler. Returns “true” if successful, “false: <i>error message</i>” otherwise where <i>error message</i> provides the message to display to the user.</p> <p>Instance id is also passed with the jobid.</p> <p>Example:</p> <p>For jobs that maintain job history:</p> <pre>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?button=remove&jobid=1001&instanceid=2</pre> <p>For jobs that do not maintain job history:</p> <pre>http://server:1741/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet?button=remove&jobid=1001&instanceid=2</pre>

Customizing the Job Browser Button Behaviors

To customize the behavior of the buttons in the Job Browser dialog box, modify the action configuration file summarized in [Table 18-5](#). The action configuration file contains the tags listed in [Table 18-6](#).

Table 18-5 Job Browser Action Configuration File

Name	JrmButtonActions.xml
Description	When the user selects a job and clicks one of the buttons in the Job Browser dialog box, JrmJobApplet uses this file to determine the URL to be called. All action URLs are invoked via an HTTP GET request.
Runtime Location	$\$NMSROOT$ /htdocs/jrm/JrmButtonActions.xml where $\$NMSROOT$ is the directory in which the product was installed.
Guidelines/Restrictions	Applications that do not want the default JRM actions must add the action URLs for their job type to this file.

CISCO CONFIDENTIAL**Table 18-6 Job Browser Action Configuration File Contents**

Tag	Attributes	Description
ACTIONS		Container for all button actions.
JOBTYPE		Container for a job type.
	ID	A string identifying the job type and subtypes (for example, SWIM:update.)
ACTION		Defines the URL that is called when the user requests an action.
	BUTTON	<p>Allowed values:</p> <ul style="list-style-type: none"> details—Invoked when the user clicks Job Details. The details button URL is displayed in a separate browser instance. remove—Invoked when the user clicks Remove Job. stop or kill—When the user clicks Stop Job, the Job Browser presents two options: <ul style="list-style-type: none"> Stop the job (finish gracefully). Kill the job unconditionally.
	URL	<p>The URL to be called to perform an action.</p> <ul style="list-style-type: none"> details—If there is no URL in the actions file for the selected job type, an error dialog box is displayed. remove—Default action is to call the JRM servlet to remove the job. stop or kill—Default action for both stop and kill is to ask the Daemon Manager to kill the job. <p>Note A stop action can be specified for a particular job type without specifying a kill action and vice versa.</p>

Return values for the BUTTON and URL tags shown in [Table 18-6](#) is as follows:

- details: Returns an application-specific HTML page that displays the job details. The application must display any error messages.
- remove, stop, kill: Returns “true” if the operation *initiated* successfully (does not mean it completed); “false:Error Message” if an error occurred. A dialog box displays the error message.

[Example 18-10](#) shows the default Job Browser action configuration file.

Example 18-10 Default Job Browser Action Configuration File

```
<?xml version="1.0"?>

<ACTIONS>
  <JOBTYPE ID="Test">
    <ACTION BUTTON="details" URL="/jrm/TestDetails.html" />
    <ACTION BUTTON="stop" URL="/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet" />
    <ACTION BUTTON="kill" URL="/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet" />
    <ACTION BUTTON="remove" URL="/CSCOnm/servlet/com.cisco.nm.cmf.jrm.JrmServlet" />
  </JOBTYPE>
</ACTIONS>
```

CISCO CONFIDENTIAL

```

<JOBTYPE ID="NetConfigJob">
  <ACTION BUTTON="stop"
URL="/CSCOnm/servlet/com.cisco.nm.config.netconfig.server.NetConfigUIServlet" />
  <ACTION BUTTON="kill"
URL="/CSCOnm/servlet/com.cisco.nm.config.netconfig.server.NetConfigUIServlet" />
  <ACTION BUTTON="remove"
URL="/CSCOnm/servlet/com.cisco.nm.config.netconfig.server.NetConfigUIServlet" />
  <ACTION BUTTON="details" URL="/netconfig/netconfig.jsp" />
</JOBTYPE>

<JOBTYPE ID="NetConfigPurge">
  <ACTION BUTTON="remove"
URL="/CSCOnm/servlet/com.cisco.nm.config.netconfig.server.NetConfigUIServlet" />
  <ACTION BUTTON="kill"
URL="/CSCOnm/servlet/com.cisco.nm.config.netconfig.server.NetConfigUIServlet" />
  <ACTION BUTTON="stop"
URL="/CSCOnm/servlet/com.cisco.nm.config.netconfig.server.NetConfigUIServlet" />
</JOBTYPE>
</ACTIONS>

```

To customize the Job Details and Stop Job buttons but rely on the default JRM action for the Remove Job button for the ACL Manager, add the following element to the action configuration file:

```

<JOBTYPE ID="acl">
  <ACTION VERB="details" URL="/acl/editjob"/>
  <ACTION Verbosity" URL="/acl/stopjob"/>
</JOBTYPE>

```

Using this action configuration, if the user selects an ACL job with Id=42 and clicks “Job Details”, the JRM browser will issue the following GET request and display the result in a new browser window:

```
http://server:1741/acl/editjob?jobid=42&button=details
```

If the user clicks “Stop Job” and selects “stop” (not “kill”) from the dialog box, the JRM browser will issue the following GET request:

```
http://server:1741/acl/stopjob?jobid=42&button=stop
```

If the response is “true,” a dialog box is displayed indicating that the operation was initiated successfully. If the response is “false: device not responding” (for example), a dialog box will be displayed with the text, “device not responding.”

Using JRM from the Command Line

JRM includes two command line applications, `jobcli` and `lockcli`, that provide a command language interface for scheduling jobs and locking resources. These applications are used for debugging purposes and to provide a JRM interface for non-Java applications such as Perl or C++.

The following topics describe `jobcli` and `lockcli`:

- [Job Command Line Interface](#)
- [Lock Command Line Interface](#)

Related Topics

See the “[Using the Job Command-Line Commands](#)” section on page 18-59

CISCO CONFIDENTIAL

Job Command Line Interface

The job command line application, `jobcli` (shown in [Table 18-7](#)), is a Java application that provides a simple job manipulation command language.

Table 18-7 Jobcli Interface

Name	com.cisco.nm.cmf.jrm.jobcli	
Description	Provides a simple command language that allows you to: <ul style="list-style-type: none"> • Create or create and run a job • Approve or reject a job • Cancel, delay, delete, suspend, or resume a job • Change job schedule You can provide inputs to <code>jobcli</code> using either: <ul style="list-style-type: none"> • Standard input • A file of commands The <code>jobcli</code> commands are described in the “Using the Job Command-Line Commands” section on page 18-59.	
Syntax	jre -cp classpath com.cisco.nm.cmf.jrm.jobcli [-f clifile]	
Arguments	Name	Description
	<i>classpath</i>	Environment variable that tells the interpreter where to look for user-defined classes.
	<i>-f clifile</i>	Reads commands from <i>clifile</i> . If <i>-f</i> option is missing, commands are read from standard input.
Outputs	Sent to stdout/stderr.	

Lock Command Line Interface

The lock command line application, `lockcli` (shown in [Table 18-8](#)), is a Java application that provides a simple lock manipulation command language.

Table 18-8 Lockcli Interface

Name	com.cisco.nm.cmf.jrm.lockcli	
Description	Provides a command language that allows you to lock and unlock a single or several resources for the owner running outside JRM. Note There is no automatic unlocking. If you lock a resource without specifying the lock duration, be sure you unlock it.	
Syntax	jre -cp classpath com.cisco.nm.cmf.jrm.lockcli {-l -u} owner resource[@duration]...	

CISCO CONFIDENTIAL**Table 18-8** *Lockcli Interface*

Arguments	Name	Description
	-l or -u	Lock or unlock resource(s).
	owner	Resource owner. Should contain at least one alphabetic character to distinguish it from jobs run under Job Browser.
	resource	Specifies resource path. If followed by <i>@duration</i> : <ul style="list-style-type: none"> • If <i>duration</i> is greater than zero, <i>resource</i> will be locked for <i>duration</i> seconds. • If <i>duration</i> is less than or equal to zero, an error message will be displayed. • If no <i>duration</i> is specified, resource will be locked until it is explicitly unlocked by its owner. <i>Duration</i> is ignored when unlocking.
Output	Success = 0 Error = >0 and stderr contains a diagnostic message.	
Examples	<p>The following example locks switch1.cisco.com for 30 seconds and switch2.cisco.com until explicitly unlocked on behalf of swim1 job:</p> <pre>jre -cp ... com.cisco.nm.cmf.jrm.lockcli -l swim1 switch1.cisco.com@30 switch2.cisco.com</pre> <p>The following example unlocks switch2.cisco.com:</p> <pre>jre -cp ... com.cisco.nm.cmf.jrm.lockcli -u swim1 switch2.cisco.com</pre>	

JRM Command Reference

JRM provides interfaces from Java, IDL, and servlets, and via command line utilities. These topics describe the reference information for these interfaces:

- [About the Job and Resource Lock Attributes](#)
- [About Displayed Job Status Values](#)
- [About the Job Manager Methods](#)
- [About the Lock Manager Methods](#)
- [About the Helper API Methods](#)
- [About the JRM Java Constants](#)
- [Using the Job Command-Line Commands](#)

About the Job and Resource Lock Attributes

[Table 18-9](#) describes the available job attributes.

CISCO CONFIDENTIAL**Table 18-9 Job Attributes**

Attribute	Description
ID	A unique number assigned to this job at creation time. This number is never reused.
InstanceID	A unique number that is incremented for every instance of job history jobs. For jobs without multiple instances, value is 0.
Type	String identifying the job type and job subtypes (for example, SWIM:update.)
Description	String that describes the job.
Command line	<p>The command line to start the job. JRM performs the following parameter substitutions on the command line:</p> <p>String/Result</p> <p>\$JC — Java classpath. Prefix with <code>-cp</code>.</p> <p>\$JI — Job ID.</p> <p>\$JJ — Sets <code>nm.jrm.jobid</code> Java property to the job ID; equivalent to <code>-Dnm.jrm.jobid=\$JI</code>.</p> <p>\$II — Instance ID.</p> <p>\$IJ — Sets <code>nm.jrm.instanceid</code> Java property to the instance ID; equivalent to <code>-Dnm.jrm.instanceid=\$II</code>.</p> <p>\$JP — Path to Java interpreter.</p> <p>\$JR — RME installation root directory.</p> <p>\$: — Path separator, the value of the <code>path.separator</code> system property (':' on UNIX, ';' on Windows).</p> <p>\$/ — File separator, the value of the <code>file.separator</code> system property ('/' on UNIX, '\' on Windows).</p>
Host	Machine name or IP address where the job will run. (For future extensions. Currently, the job is always started on the local machine.)
Account	Account under which the job is run. (For future extensions.)
Schedule	How often this job will run. Options include: run immediately, run once, run on a calendar basis (periodic), run on a time-start basis, or run on a time-stop basis.
Dependencies	A list of the Job IDs that must complete successfully. (Not currently implemented.)
Completion state	Describes the current state or last run result of the job. Job states include: running, never, suspended, wait for approval, scheduled (pending), rescheduled, completed succeeded, failed, crashed, canceled, rejected, or ERROR.
Schedule state	Determines if the job can be scheduled to run based on whether it is enabled, requires approval, or has already been approved.
Start and stop times from last run	Time stamps from the last time the job was run or attempted to run.
Progress status	Updates or diagnostic information.
Reference	An application-specific string. May contain the URL of job results.
Owner	Account of the person that created the job.
Creation time	Time the job was created.
Last modification time	Time the job was last modified.
Approver	Account of the approver. Valid only if approval is required.

CISCO CONFIDENTIAL

Table 18-10 summarizes the available resource lock attributes.

Table 18-10 Resource Lock Attributes

Field	Description
Resource path	String defining the device name and any subnode.
Owner	Job ID represented as a string.
Time stamp	Time the lock was established.
Expiration time	Time the lock expires.

About Displayed Job Status Values

Displayed job status value vary according to how often the job is run and whether approval is required.

Table 18-11 summarizes the displayed job status for run-once, approval-required jobs.

Table 18-11 Run-Once Approval-Required Job Status Values

Schedule vs. Current Time	Run State	Enabled	Approval State	Displayed Schedule Status	Displayed Completion Status
Future	Never	N		Suspended	
Future	Never	Y	Wait for approval	Wait for approval	
Future	Never	Y	Approved	Scheduled (pending)	
Future	Never	Y	Rejected	Rejected	
Future	Canceled			Canceled	
Past	Never	N			Suspended
Past	Never	Y	Wait for approval — rejected		Rejected
Past	Never	Y	Approved		ERROR
Past	Canceled				Canceled
Past	All others				Same as run state

Table 18-12 summarizes the displayed job status for run-once, no-approval-required jobs.

Table 18-12 Run-Once No-Approval Job Status Values

Schedule vs. Current Time	Run State	Enabled	Displayed Schedule Status	Displayed Completion Status
Future	Never	N	Suspended	
Future	Never	Y	Scheduled (pending)	
Future	Canceled		Canceled	

CISCO CONFIDENTIAL**Table 18-12** Run-Once No-Approval Job Status Values (continued)

Past	Never	N	Suspended
Past	Never	Y	ERROR
Past	Canceled		Canceled
Past	All others		Same as run state

Table 18-13 summarizes the displayed job status for periodic, approval-required jobs.

Table 18-13 Periodic Approval-Required Job Status Values

Run State	Enabled	Approval State	Displayed Schedule Status	Displayed Completion Status
*	N		Suspended	Same as Run state
Canceled	Y		Canceled	Canceled
All others	Y	Wait for approval	Wait for approval	Same as Run state
All others	Y	Approved	Scheduled (pending)	Same as Run state
All others	Y	Rejected	Rejected	Same as Run state

Table 18-14 summarizes the displayed job status for periodic, no-approval-required jobs.

Table 18-14 Periodic No-Approval Job Status Values

Run State	Enabled	Displayed Schedule Status	Displayed Completion Status
	N	Suspended	Same as Run State
Canceled	Y	Canceled	Canceled
All others	Y	Scheduled	Same as Run State

About the Job Manager Methods

Use the Job Manager methods summarized in Table 18-15 to add JRM scheduling functionality to your application. These methods return Java constants described in the “About the JRM Java Constants” section on page 18-56.

Table 18-15 Job Manager Method Summary

Returns	Syntax and Description
int	<code>job_cancel(int idJob);</code> Cancels a running job
int	<code>job_cancel_instance(int idJob, int instanceId, boolean cancelAllInstances);</code> Cancels a running job with instance ID.
int	<code>job_cancel_event(int idJob);</code> Cancels a running event

CISCO CONFIDENTIAL**Table 18-15 Job Manager Method Summary (continued)**

Returns	Syntax and Description
int	job_cancel_instance_event (int idJob, int instanceId, boolean cancelAllInstances); Cancels a running job with instance ID, and specified whether to cancel the instance alone or the entire job
int	job_create (JobInfo job_info, org.omg.CORBA.IntHolder idJob); Creates a job
int	job_create_hist (JobInfo jiJob, org.omg.CORBA.IntHolder idJob); Creates a job with job history
int	job_delete (int idJob); Deletes a job
int	job_delete_instance (int idJob, int instanceId, boolean delFlag); Deletes a job with the given id, instance id.
int	job_enum (JobIterHolder job_iter); Creates a job enumerator
int	job_enum_hist (JobIterHist job_iter); Creates a job enumerator
int	job_get_info (int idJob, JobInfoHolder job_info); Gets information about a job
int	job_get_info_hist (int idJob, int instanceId, JobInfoHistHolder jiJobHist); Adds job information history about a job
int	job_get_result (int idJob, IntHolder status); Gets job run state
int	job_get_schedule (int idJob, ScheduleHolder schedule); Fills schedule with job schedule
int	job_get_schedule_string (int idJob, StringHolder schedule); Gets job schedule information
int	job_run (int idJob); Runs a job immediately
int	job_set_approved (int idJob, boolean bApproved, String szApprover) Approves or rejects a job
int	job_set_info (JobInfo job_info); Updates job information
int	job_set_info_hist (JobInfoHist jiJobHist); Updates job information
int	job_set_progress_string (int idJob, String szStatus); Sets progress string
int	job_set_reference (int idJob, String szReference); Sets job reference attribute
int	job_set_result (int idJob, int state); Sets job run state
int	job_set_resume (int idJob, boolean bResume); Enables or disables a job
int	job_set_schedule (int idJob, Schedule schedule); Sets job schedule
int	next (JobInfoHolder job_info); Fills job_info with job information

CISCO CONFIDENTIAL**Table 18-15** Job Manager Method Summary (continued)

Returns	Syntax and Description
int	<code>next_n(int max_jobs, JobInfoSequenceHolder job_seq);</code> Fills job_seq with job descriptions
int	<code>release();</code> Releases an iterator

job_cancel

```
int job_cancel (int idJob);
```

Cancels a job if it is running. The job sends a request to stop.

**Note**

JRM only issues the request. It does not wait until the process actually stops.

Input Arguments

idJob [int] Unique number assigned to a job at creation time.

Return Values

STATUS_Ok Job was canceled or is not running.

STATUS_NotFound No such job.

Usage Guidelines

To cancel a running job, send a request to the Daemon Manager to stop the process.

job_cancel_instance

```
int job_cancel_instance (int idJob, int instanceId, boolean cancelAllInstances);
```

Cancels an instance job if it is running.

**Note**

JRM only issues the request. It does not wait until the process actually stops.

Input Arguments

idJob [int] Unique number assigned to a job at creation time.

idInstance [int] Unique number assigned to an instance at creation time.

cancelAllInstances [boolean] Indicates whether you want to cancel all future instances or just this instance

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok	Job instance was canceled or is not running.
STATUS_NotFound	No such instance.

Usage Guidelines

To cancel a running job, send a request to the Daemon Manager to stop the process.

job_cancel_event

```
int job_cancel_event (int idJob)
```

Cancels a job if it is running. Sends a cancel event to the running job. The job should process the event and stop the event by itself.

**Note**

JRM only issues the request. It does not wait until the process actually stops.

Input Arguments

idEvent	[int] Unique number assigned to an event at creation time.
---------	--

Return Values

STATUS_Ok	Event was canceled or is not running.
STATUS_NotFound	No such Event.

Usage Guidelines

To cancel a running job, send a request to the Daemon Manager to stop the process.

job_cancel_instance_event

```
int job_cancel (int idJob, int instanceId, boolean cancelAllInstances);
```

Cancels an instance of an event if it is running.

**Note**

JRM only issues the request. It does not wait until the process actually stops.

CISCO CONFIDENTIAL**Input Arguments**

<code>idJob</code>	[int] Unique number assigned to a job at creation time.
<code>instanceId</code>	[int] Unique number assigned to a job instance at creation time.
<code>cancelAllInstances</code>	[boolean] Indicates whether you want to cancel all future instances or just this instance

Return Values

<code>STATUS_Ok</code>	Instance of the job was canceled or is not running.
<code>STATUS_NotFound</code>	No such job instance.

Usage Guidelines

To cancel a running instance of the job, send a request to the Daemon Manager to stop the process.

job_create

```
int job_create (JobInfo job_info, org.omg.CORBA.IntHolder idJob);
```

Creates a job. The ID field and all fields related to the last job execution are ignored.

Input Arguments

<code>job_info</code>	[JobInfo] Job information. The JobInfo structure is defined in the IDL file (see the “About the IDL Interface” section on page 18-8).
-----------------------	---

Output Arguments

<code>id_Job</code>	[org.omg.CORBA.IntHolder] Unique number assigned to a job at creation time.
---------------------	---

Return Values

<code>STATUS_Ok</code>	Success. On return, <code>idJob</code> contains the unique job ID.
<code>STATUS_NotFound</code>	Job not found.

job_create_hist

```
int job_create_hist (JobInfo jiJob, org.omg.CORBA.IntHolder idJob);
```

Creates a job. The ID field and all fields related to the last job execution are ignored.

CISCO CONFIDENTIAL**Input Arguments**

`job_info` [JobInfo] Job information. The JobInfo structure is defined in the IDL file (see the “About the IDL Interface” section on page 18-8).

Output Arguments

`id_Job` [org.omg.CORBA.IntHolder] Unique number assigned to a job at creation time.

Return Values

`STATUS_Ok` Success. On return, `idJob` contains the unique job ID.

`STATUS_NotFound` Job not found.

job_delete

```
int job_delete (int idJob);
```

Deletes the job with the given ID.

Input Arguments

`idJob` [int] Unique number assigned to a job at creation time.

Return Values

`STATUS_Ok` Success.

`STATUS_NotFound` No such job.

job_delete_instance

```
int job_delete_instance (int idJob, int instanceId, boolean delFlag);
```

Deletes the job with the given ID.

Input Arguments

`idJob` [int] Unique number assigned to a job at creation time.

`instanceId` [int] Unique number assigned to an instance of a job at creation time.

`delFlag` [boolean] Indicates whether you want to delete all instances or just this instance.

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok	Success.
STATUS_NotFound	No such job.

job_enum

```
int job_enum (JobIterHolder job_iter);
```

Creates the job enumerator.

Output Arguments

job_iter	[JobIterHolder] Object used to retrieve the next job.
----------	---

Return Values

STATUS_Ok	Success.
-----------	----------

Example

Use this method with the next and *release* methods to retrieve the next job.

```
JobIterHolder jih = new JobIterHolder ();
JobInfoHolder jobinfo = new JobInfoHolder ();
/* Get the JobIter and browse through it */
if (STATUS_Ok == job_manager.job_enum(jih))
{
    while (STATUS_Ok == jih.value.next(jobinfo))
    {
        // do the required operations on JobInfo
    }
}
//Calling the release of JobIter
jih.value.release();
```

The functions next (), next_n() and release() are to be called on the JobIter reference, which can be obtained by calling the job_enum API.

job_enum_hist

```
int job_enum_hist (JobIterHist job_iter);
```

Creates the job enumerator.

Output Arguments

job_iter	[JobIterHolder] Object used to retrieve the next job.
----------	---

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok Success.

Example

Use this method with the next and *release* methods to retrieve the next job.

```
JobIterHolder jih = new JobIterHolder ();
JobInfoHolder jobinfo = new JobInfoHolder ();
/* Get the JobIter and browse through it */
if (STATUS_Ok == job_manager.job_enum(jih))
{
    while (STATUS_Ok == jih.value.next(jobinfo))
    {
        // do the required operations on JobInfo
    }
}
//Calling the release of JobIter
jih.value.release();
```

The functions next (), next_n() and release() are to be called on the JobIter reference, which can be obtained by calling the job_enum API.

job_get_info

```
int job_get_info (int idJob, JobInfoHolder job_info);
```

Fills the job information data structure with information about a given job.

Input Arguments

idJob [int] Unique number assigned to a job at creation time.

Output Arguments

job_info [JobInfoHolder] Job information. The JobInfo structure is defined in the IDL file (see the [“About the IDL Interface”](#) section on page 18-8).

Return Values

STATUS_Ok Success.

STATUS_NotFound No such job.

job_get_info_hist

```
int job_get_info (int idJob, int instanceId, JobInfoHistHolder jiJobHist);
```

Fills the job information data structure with information about a given job.

CISCO CONFIDENTIAL**Input Arguments**

<code>idJob</code>	[int] Unique number assigned to a job at creation time.
<code>instanceId</code>	[int] Unique number assigned to an instance of a job at creation time.

Output Arguments

<code>job_info</code>	[JobInfoHolder] Job information. The JobInfo structure is defined in the IDL file (see the “ About the IDL Interface ” section on page 18-8).
-----------------------	---

Return Values

<code>STATUS_Ok</code>	Success.
<code>STATUS_NotFound</code>	No such job.

job_get_result

```
int job_get_result (int idJob, IntHolder status);
```

Retrieves the current run state of a job.

Input Arguments

<code>idJob</code>	[int] Unique number assigned to a job at creation time.
--------------------	---

Output Arguments

<code>status</code>	[IntHolder] Current run state.
---------------------	--------------------------------

Return Values

<code>STATUS_Ok</code>	Success.
<code>STATUS_NotFound</code>	No such job.

job_get_schedule

```
int job_get_schedule (int idJob, ScheduleHolder schedule);
```

Fills the schedule data structure with the job’s scheduling information.

CISCO CONFIDENTIAL**Input Arguments**

`idJob` [int] Unique number assigned to a job at creation time.

Output Arguments

`schedule` [ScheduleHolder] Job scheduling information. The Schedule structure, which is defined in the IDL file (see the “[About the IDL Interface](#)” section on page 18-8), includes the next time to start, the type of schedule, and the time increment.

Return Values

`STATUS_Ok` Success.

`STATUS_NotFound` No such job.

job_get_schedule_string

```
int job_get_schedule_string (int idJob, StringHolder schedule);
```

Puts a displayable representation of the job schedule into the string contained in `schedule`.

Input Arguments

`idJob` [int] Unique number assigned to a job at creation time.

Output Arguments

`schedule` [StringHolder] Displayable representation of the job’s schedule.

Return Values

`STATUS_Ok` Success.

`STATUS_NotFound` No such job.

job_run

```
int job_run (int idJob);
```

Runs the job immediately.

Input Arguments

`idJob` [int] Unique number assigned to a job at creation time.

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok	Success.
STATUS_NotFound	No such job.

job_set_approved

```
int job_set_approved (int idJob,
                    boolean bApproved,
                    String szApprover)
```

Approves or rejects a job. This method approves or rejects a job and records the approver name.

Input Arguments

idJob	[int] Unique number assigned to a job at creation time.
bApproved	[boolean] True = approve job. False = reject job.
szApprover	[String] Account of the approver.

Return Values

STATUS_Ok	Success.
STATUS_NotFound	No such job.

Usage Guidelines

If a nonperiodic job that requires approval has not been approved by the time it is scheduled to run, it is automatically rejected.

job_set_info

```
int job_set_info (JobInfo job_info);
```

Replaces all job information.

Input Arguments

job_info	[JobInfo] Job information. The JobInfo structure is defined in the IDL file (see the “About the IDL Interface” section on page 18-8).
----------	---

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok	Success.
STATUS_NotFound	No such job.

job_set_info_hist

```
int job_set_info_hist (JobInfoHist jiJobHist);
```

Replaces all job information.

Input Arguments

job_info [JobInfo] Job information. The JobInfo structure is defined in the IDL file (see the [“About the IDL Interface”](#) section on page 18-8).

Return Values

STATUS_Ok	Success.
STATUS_NotFound	No such job.

job_set_progress_string

```
int job_set_progress_string (int idJob, String szStatus);
```

Sets the progress string with update or diagnostic information.

Input Arguments

idJob [int] Unique number assigned to a job at creation time.

szStatus [String] Updates or diagnostic information.

Return Values

STATUS_Ok	Success.
STATUS_NotFound	No such job.

job_set_reference

```
int job_set_reference (int idJob, String szReference);
```

Sets the job’s reference attribute.

CISCO CONFIDENTIAL**Input Arguments**

<code>idJob</code>	[int] Unique number assigned to a job at creation time.
<code>szReference</code>	[String] An application-specific string. May contain the URL of the job results.

Return Values

<code>STATUS_Ok</code>	Success.
<code>STATUS_NotFound</code>	No such job.

job_set_result

```
int job_set_result (int idJob, int state);
```

Sets the job's current run state. The only states the application is allowed to set are Succeeded, SucceededWithInfo, or Failed.

Input Arguments

<code>idJob</code>	[int] Unique number assigned to a job at creation time.
<code>state</code>	[int] Current run state.

Return Values

<code>STATUS_Ok</code>	Success.
<code>STATUS_NotFound</code>	No such job.
<code>STATUS_BadArgument</code>	State was not Succeeded, SucceededWithInfo, Never Ran, Canceled, CanceledInstance, or Failed

job_set_resume

```
int job_set_resume (int idJob, boolean bResume);
```

Resumes or suspends a job. When a previously suspended job is resumed, it is scheduled to run according to its schedule type (run once or periodic) provided that it is approved or does not require approval.

Input Arguments

<code>idJob</code>	[int] Unique number assigned to a job at creation time.
<code>bResume</code>	[boolean] True = resume job. False = suspend job.

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok	Success.
STATUS_NotFound	No such job.

Usage Guidelines

You can use the following technique when a job needs to be run immediately but only after certain actions are performed by the job creator:

- Create a job with schedule specifying to run it immediately but in the suspended state. You now have a job ID.
- Perform whatever actions are needed that reference job ID.
- Enable (resume) the job. If approved, the job will run immediately.

job_set_schedule

```
int job_set_schedule (int idJob, Schedule schedule);
```

Sets the job's schedule to **schedule**.

Input Arguments

idJob	[int] Unique number assigned to a job at creation time.
schedule	[Schedule] Job scheduling information. The Schedule structure, which is defined in the IDL file (see the “About the IDL Interface” section on page 18-8), includes the next time to start, the type of schedule, and the time increment.

Return Values

STATUS_Ok	Success.
STATUS_NotFound	No such job.

next

```
int next (JobInfoHolder job_info);
```

Returns the JobInfo instance for the next task entry.

Output Arguments

job_info	[JobInfoHolder] Job information. The JobInfo structure is defined in the IDL file (see the “About the IDL Interface” section on page 18-8).
----------	--

CISCO CONFIDENTIAL**Return Values**

STATUS_OK	Filled <i>job_info</i> .
STATUS_EOF	No more entries.

Example

See the “[job_enum](#)” section on page 18-35.

next_n

```
int next_n (int max_jobs, JobInfoSequenceHolder job_seq);
```

Fills the job holder array with the next group of jobs.

Input Arguments

max_jobs [int] Maximum number of jobs that can be returned in the job holder array.

Output Arguments

job_seq [JobInfoSequenceHolder] An array of objects that allows you to retrieve the next *max_jobs* jobs.

Return Values

STATUS_OK	Put at least one element into <i>job_seq</i> .
STATUS_EOF	No more jobs.

Example

See the “[job_enum](#)” section on page 18-35.

release

```
int release();
```

Releases the iterator and makes it unavailable to the clients.

Arguments

None

Example

See the “[job_enum](#)” section on page 18-35.

CISCO CONFIDENTIAL**About the Lock Manager Methods**

Use the Lock Manager methods summarized in Table to add JRM resource locking functionality to your application. These methods return Java constants described in the [“About the JRM Java Constants”](#) section on page 18-56.

Table 18-16 JRM Lock Manager Method Summary

Returns	Syntax and Description
int	<code>enum_job_locks</code> (String szJob, LockIterHolder lock_iter); Creates an iterator
int	<code>find_lock</code> (String szResource, LockInfoHolder lock_info); Finds a lock entry
int	<code>get_lock</code> (String szResource, LockInfoHolder lock_info); Gets lock information
int	<code>lock</code> (String szResource, String szOwner, int duration); Locks a resource
int	<code>lock_n</code> (LockRequest[] Locks, String szOwner); Locks multiple resources
int	<code>next</code> (LockInfoHolder lock_info); Fills <i>lock_info</i>
int	<code>next_n</code> (int max_locks, LockInfoSequenceHolder lock_seq); Fills <i>lock_seq</i>
int	<code>release</code> (); Releases an iterator
int	<code>unlock</code> (String szResource, String szOwner); Unlocks a resource
int	<code>unlock_job</code> (String szJob); Unlocks all locks for a job
int	<code>unlock_n</code> (String[] szResource, String szOwner); Unlocks multiple resources

enum_job_locks

Status `enum_job_locks` (String szJob, LockIterHolder lock_iter);

Creates an iterator for all the locks for this job or process.

Input Arguments

szJob [String] For a job, the string representation of the job ID.
For a process, the name known to the Daemon Manager.

Output Arguments

lock_iter [LockIterHolder] Object used to retrieve the next lock.
The LockIter structure is defined in the IDL file (see the [“About the IDL Interface”](#) section on page 18-8).

CISCO CONFIDENTIAL**Return Values**

STATUS_Ok	Success. <i>lock_iter</i> is returned.
STATUS_NotFound	No such owner.

Example

Use this method with the *next* and *release* methods to retrieve the next lock.

```
LockIterHolder lih = new LockIterHolder ();
LockInfoHolder lockInfo = new LockInfoHolder ();
/* Get the LockIter and browse through it */
if (STATUS_Ok == lock_manager.enum_job_locks(lih))
{
    while (STATUS_Ok == lih.value.next(lockInfo))
    {
        // do the required operations on LockInfo
    }
}
//Calling the release of LockIter
lih.value.release();
```

find_lock

```
Status find_lock (String szResource, LockInfoHolder lock_info);
```

Finds the lock entry that *prevents* a device from being locked. Unlike *get_lock*, which returns the lock information for a specific device, *find_lock* returns the lock information for the device that is preventing another resource from being locked.

For more information about the locking hierarchy, see the “[Locking Parts of a Device](#)” section on [page 18-5](#).

Input Arguments

szResource [String] Device name and any subnode.

Output Arguments

lock_info [LockInfoHolder] Lock information. The LockInfo structure is defined in the IDL file (see the “[About the IDL Interface](#)” section on [page 18-8](#)).

Return Values

STATUS_Ok	Success.
STATUS_NotFound	No such resource.

get_lock

```
Status get_lock(String szResource, LockInfoHolder lock_info);
```

CISCO CONFIDENTIAL

Returns lock information for a device. This method differs from `find_lock`, which finds the lock entry that is *preventing* a device from being locked.

Input Arguments

`szResource` [String] Device name and any subnode.

Output Arguments

`lock_info` [LockInfoHolder] Lock information. The `LockInfo` structure is defined in the IDL file (see the [“About the IDL Interface”](#) section on page 18-8).

Return Values

`STATUS_Ok` Success.

`STATUS_NotFound` No such resource.

lock

```
Status lock (String szResource, String szOwner, int duration);
```

Locks the resource for *duration* seconds. If the job already owns this resource, this method will change the lock expiration time.

Input Arguments

`szResource` [String] Device name and any subnode.

`szOwner` [String] For a job, the string representation of the resource owner. For a process, the name known to the Daemon Manager.

`duration` [int] Time (seconds) for which the resource is to be locked.

Return Values

`STATUS_Ok` Success (job was killed or is not running).

`STATUS_Exists` The lock for that resource already exists.

lock_n

```
Status lock_n (LockRequest[] Locks, String szOwner);
```

Locks multiple resources.

CISCO CONFIDENTIAL**Input Arguments**

<code>locks</code>	[LockRequest] An array of objects that allows you specify the resources to be locked.
<code>szOwner</code>	[String] For a job, the string representation of the resource owner. For a process, the name known to the Daemon Manager.

Return Values

<code>STATUS_Ok</code>	Success. (<i>All resources have been successfully locked.</i>)
<code>STATUS_Exists</code>	At least one resource could not be locked.

next

```
Status next (LockInfoHolder lock_info);
```

Fills the lock holder with the lock entry information and advances to the next lock entry in the locks list.

Output Arguments

`lock_info` [LockInfoHolder] Lock information. The LockInfo structure is defined in the IDL file (see the [“About the IDL Interface”](#) section on page 18-8).

Return Values

<code>STATUS_Ok</code>	Success (job was killed or is not running).
<code>STATUS_EOF</code>	End of iteration.

Example

See the [“enum_job_locks”](#) section on page 18-44.

next_n

```
Status next_n (int max_locks, LockInfoSequenceHolder lock_seq);
```

Fills the lock holder array with the next group of locks.

Input Arguments

`max_locks` [int] Maximum number of locks that can be returned in LockInfoSequence.

CISCO CONFIDENTIAL**Output Arguments**

`lock_seq` [LockInfoSequenceHolder] An array of objects that allows you to retrieve the next *max_locks* locks. The LockInfo structure is defined in the IDL file (see the [“About the IDL Interface”](#) section on page 18-8).

Return Values

`STATUS_OK` Success (at least one element put into *lock_seq*).

`STATUS_EOF` No more elements.

Example

See the [“enum_job_locks”](#) section on page 18-44.

release

```
Status release();
```

The **release** method releases an iterator and makes it unavailable to the clients.

Arguments

None

Example

See the [“enum_job_locks”](#) section on page 18-44.

unlock

```
Status unlock (String szResource, String szOwner);
```

Unlocks the specified resource.

Input Arguments

`szResource` [String] Name of the resource to be unlocked.

`szOwner` [String] For a job, the string representation of the resource owner. For a process, the name known to the Daemon Manager.

Return Values

`STATUS_OK` Success.

`STATUS_NotFound` Resource was not locked.

CISCO CONFIDENTIAL

unlock_job

Status `unlock_job` (String szJob);

Release all locks for the specified job.

Input Arguments

szJob [String] For a job, the string representation of the job ID. For a process, the name known to the Daemon Manager.

Return Values

STATUS_Ok Success. Resources released successfully or there were no resources locked by this job.

unlock_n

Status `unlock_n` (String[] Resource, String szOwner);

Unlocks all the resources in the specified list.

Input Arguments

Resource [String] Array of device names and any subnodes.

szOwner [String] For a job, the string representation of the resource owner. For a process, the name known to the Daemon Manager.

Return Values

STATUS_Ok Success. Released resources.

STATUS_NotFound At least one resource was not locked.

About the Helper API Methods

The Helper API consists of the class Client and the inner class of Client, MyJob. Only jobs running under JRM can use the helper methods in the MyJob class. These methods return Java constants described in the [“About the JRM Java Constants” section on page 18-56](#).

Related Topics

- [About the Helper API](#)
- [About the JRM Java Constants](#)
- [Parsing ESS Messages](#)

CISCO CONFIDENTIAL

Location com.cisco.nm.cmf.jrm.Client

Client Class Constructor Summary

public class **Client** implements Constants

This class is a collection of helper functions that can be called by JRM clients. It contains two groups of functions:

- Those usable by any client.
 - Those usable only by jobs running under JRM.
-

Client Class Method Summary

Returns	Syntax and Description
static String	getScheduleString (Schedule sch) Returns a printable representation of the schedule string. Defines SCHTYPE constants.
static void	getStateStrings (JobInfo ji, StringHolder h_szRunState, StringHolder h_szSchState) Returns a printable representation of a job's run and schedule states.
Properties	getOrbConnectionProperties () Initializes the ORB and locates servers.

MyJob Class Constructor Summary

public static class **MyJob**

This inner class is a collection of the static methods that can be used only from jobs running under JRM.

The methods in MyJob automatically establish connection with ORB. They obtain the value of Job Id (which they need to communicate to the JRM) from the nm.jrm.jobid property.

The easiest way to set this property is to add the \$JJ parameter to the job's command line (see the [“About the Job and Resource Lock Attributes”](#) section on page 18-26).

Table 18-17 MyJob Class Method Summary

Returns	Syntax and Description
int	get_job_id (); Gets the job ID
int	get_job_instance_id (); Gets the job instance ID
int	get_job_info (JobInfoHolder h_ji); Fills h_ji with job information
int	get_job_info_hist (JobInfoHistHolder h_ji); Fills h_ji with job information with additional parameters for req_hist and instance_id

CISCO CONFIDENTIAL**Table 18-17 MyJob Class Method Summary (continued)**

Returns	Syntax and Description
int	<code>get_lock_info</code> (string szLockPath, LockInfoHolder h_li); Fills h_li with lock information
boolean	<code>is_server_running</code> (); Checks server status
int	<code>lock</code> (string szLockPath, int duration); Locks the resource for the current job
int	<code>lock_n</code> (LockRequestSequence Locks); Locks multiple resources
int	<code>set_completion_state</code> (int run_state); Sets the running job's status
int	<code>set_progress</code> (string szProgress); Sets the running job's progress string
void	<code>unlock</code> (string szLockPath); Unlocks the resource
int	<code>unlock_all</code> (); Unlocks all resources for the current job

get_job_id

```
static int get_job_id();
```

MyJob method returns the job ID by retrieving the value of nm.jrm.jobid property. This property is set by adding \$JJ on the job's command line.

Arguments

None

Return Values

0	Called outside the running job.
an integer	Job ID.

get_job_instance_id

```
static int get_job_instance_id();
```

MyJob method returns the job instance ID by retrieving the value of nm.jrm.jobinstanceid property. This property is set by adding \$JJ on the job's command line.

Arguments

None

CISCO CONFIDENTIAL**Return Values**

0 Called outside the running job.

an integer Job Instance ID.

get_job_info

```
static int get_job_info (JobInfoHolder h_ji);
```

MyJob method sets h_ji.value to JobInfo of self.

Output Arguments

h_ji [JobInfoHolder] Contains job information.

Return Values

STATUS_Ok Success.

STATUS_NotFound Not called from job.

get_job_info_hist

```
static int get_job_info_hist (JobInfoHistHolder h_ji);
```

MyJob method sets h_ji.value to JobInfoHist of self.

Output Arguments

h_ji [JobInfoHistHolder] Contains job history information.

Return Values

STATUS_Ok Success.

STATUS_NotFound Not called from job.

get_lock_info

```
static int get_lock_info (string szLockPath, LockInfoHolder h_li);
```

MyJob method that returns lock information for this lock if the resource **szLockPath** is locked.

Input Arguments

szLockPath [string] Device name and any subnode.

CISCO CONFIDENTIAL**Output Arguments**

h_li [LockInfoHolder] Contains lock information.

Return Values

STATUS_Ok Success.

STATUS_NotFound Lock not found.

getOrbConnectionProperties

```
static public Properties getOrbConnectionProperties ()
```

Initializes the ORB and locates servers.

Arguments

None

Returns

Properties [Properties] Server properties (host, port number)

getScheduleString

```
static String getScheduleString (Schedule sch)
```

Returns a printable representation of the schedule string. Defines SCHTYPE constants.

Input Arguments

sch [Schedule] Schedule object, which includes:

- sch.start—Next time to start
- sch.increment—Increment amount

Returns

szFormat [String] Printable schedule string

getStateStrings

```
static void getStateStrings (JobInfo ji, StringHolder h_szRunState, StringHolder  
h_szSchState)
```

Returns a printable representation of a job's run and schedule states.\

CISCO CONFIDENTIAL**Input Arguments**

ji [JobInfo] JobInfo structure

Output Arguments

h_szRunState [StringHolder] Run state

h_szSchState [StringHolder] Schedule state

is_server_running

```
static boolean is_server_running();
```

MyJob method that checks to see if the server is running.

Arguments

None

Return Values

True Server is running.

False Server is not running.

lock

```
static int lock (string szLockPath, int duration);
```

MyJob method that locks the resource for the current job for **duration** seconds.

Input Arguments

szLockPath [string] Device name and any subnode.

duration [int] Number of seconds to lock the resource.

Return Values

STATUS_Ok Success.

STATUS_NotFound Not called from job.

STATUS_Exists szLockPath cannot be locked.

CISCO CONFIDENTIAL

lock_n

```
static int lock_n (LockRequestSequence Locks);
```

MyJob methods that locks multiple resources.

Input Arguments

Locks	[LockRequestSequence] An array of objects that allows you to specify the resources to be locked.
-------	--

Return Values

STATUS_Ok	Success.
STATUS_NotFound	Not called from job.
STATUS_Exists	At least one resource cannot be locked.

set_completion_state

```
static int set_completion_state (int run_state);
```

MyJob method that sets the running job's status (completed successfully, failed, canceled).

Input Arguments

run_state	[JobRunState] Current run state.
-----------	----------------------------------

Return Values

STATUS_Ok	Success. If some of the resources were not locked, they are ignored.
STATUS_NotFound	Not called from job.

set_progress

```
static int set_progress (string szProgress);
```

MyJob method that sets the running job's progress string.

Input Arguments

szProgress	[string] Updates or diagnostic information.
------------	---

CISCO CONFIDENTIAL

Return Values

- STATUS_Ok Success.
- STATUS_NotFound Not called from job.

unlock

```
static void unlock (string szLockPath);
```

MyJob method that unlocks a resource.

Input Arguments

- szLockPath [string] Device name and any subnode.

Return Values

- STATUS_Ok Success.
- STATUS_NotFound No such job or no such lock.

unlock_all

```
static int unlock_all();
```

MyJob method that releases all the resources for the current job.

Arguments

None

Return Values

- STATUS_Ok Success.
- STATUS_NotFound Not called from job.

About the JRM Java Constants

This section describes the symbolic constants for Java applications. These constants are initialized in the IDL file (see the [“About Displayed Job Status Values”](#) section on page 18-28).

Table 18-18 JRM Java Method Return Codes

Constant	Description
STATUS_Ok	Success

CISCO CONFIDENTIAL**Table 18-18 JRM Java Method Return Codes**

STATUS_Exists	Entry already exists
STATUS_NotFound	Entry not found
STATUS_EOF	End of iteration

Table 18-19 JRM Job Completion States

Constant	Description
RUNST_NeverRan	The task was never run
RUNST_Running	The task is currently running
RUNST_Succeeded	Task completed successfully
RUNST_SucceededWithInfo	Task completed successfully, returning information
RUNST_Failed	Task ran and failed
RUNST_Crashed	Crashed (“core dump”)
RUNST_LaunchFailed	Job Manager could not start the task for this job
RUNST_Canceled	Canceled by client
RUNST_CanceledInstance	Canceled Instance by client

Table 18-20 JRM Schedule State Bits

Constant	Description
SCHST_RequiresApproval	Set if job requires approval.
SCHST_Enabled	Job is enabled.
SCHST_AM_Mask	Mask for the job approval state. Use (schedule_state & SCHST_AM_Mask) to compare with values below.
SCHST_AM_Waiting	Waiting for approval.
SCHST_AM_Approved	Approved.
SCHST_AM_Rejected	Rejected.

Table 18-21 JRM Schedule Types

Constant	Description
SCHTYPE_Immediate	Run job immediately
SCHTYPE_Once	Run job once
SCHTYPE_Daily	Run every <i>n</i> days
SCHTYPE_Weekly	Run every <i>n</i> weeks ¹
SCHTYPE_Monthly	Run every <i>n</i> months ²
SCHTYPE_MonthLastDay	Run on the last day of the month every <i>n</i> months

CISCO CONFIDENTIAL**Table 18-21 JRM Schedule Types**

SCHTYPE_MonthSameXday	Run on the given day (Sunday/Monday/...) of the first/second/... week of the month every n months ³
SCHTYPE_MonthLastXday	Run on the given day of the last week of the month every n months ⁴
SCHTYPE_S_Seconds	Run every n seconds
SCHTYPE_S_Minutes	Run every n minutes
SCHTYPE_S_Hours	Run every n hours
SCHTYPE_E_Seconds	Run n seconds after the previous run ended
SCHTYPE_E_Minutes	Run n minutes after the previous run ended
SCHTYPE_E_Hours	Run n hours after the previous run ended

1. Start date day of the week.
2. Start date day of the month.
3. Start date week number and the week day.
4. Start date week day.

**Note**

Some calendar options can produce impossible values (for example, run on the 31st of every month or on the 5th Friday of every month). Those impossible dates will be skipped. For example, the job scheduled to run on the 31st of the month will run only for the months that have 31 days.

Parsing ESS Messages

Use the helper class `EssMessageCreator` to parse and read the variables in the ESS message. After reading the message, your application can create an `EssMessageCreator` object using the constructor `EssMessageCreator(String message)`. This will parse the details in the message. Your application can then get the values for variables using the member variable of the object. The member variables are shown in [Table 18-22](#).

Table 18-22 ESS Member Variables

Member Variable	Description
Public String Action;	Provides commands such as start, end, etc.
public String szProgress;	Job progress status for all events,. For approve and reject events, this contains approver comments.
public int idJob;	Contains the Job ID.
public int rc;	Return code for some jobs.
public int signalNo;	Contains the signal number for daemon jobs,
public int runState;	Contains the run state value.
public String resource;	Identifies the locked resource.
public String owner;	Identifies the owner who locked or unlocked the resource.
public int instanceid;	Stores the instance ID for job-history jobs.

CISCO CONFIDENTIAL

Using the Job Command-Line Commands

Use `jobcli`, the job command-line application, to run JRM functions. [Table 18-23](#) summarizes the `jobcli` commands.

Related Topics

- [Understanding the JRM Architecture](#), page 18-5
- [Using JRM from the Command Line](#), page 18-24

Table 18-23 *jobcli* Command Summary

Syntax and Description
approve <code>jobId approver</code> Approves a job
cancel <code>jobId</code> Cancels a job
create <code>cmd=command [,descr=description] [,owner=user] [,type=string] [, schedule]</code> Creates a job
delay <code>cmd=command [,descr=description] [,owner=user] [,type=string]</code> Creates and suspends a job
delete <code>jobId</code> Deletes a job
reject <code>jobId rejecter</code> Rejects a job
resume <code>jobId</code> Resumes a job
run <code>cmd=command [,descr=description] [,owner=user] [,type=string]</code> Creates and runs a job
schedule <code>jobId schedule</code> Reschedules a job
suspend <code>jobId</code> Suspends a job
getnextschedule <code>jobId</code> Prints the next scheduled run time for the job details on clicking on the command link

approve

approve `jobId approver`

Approves the job **jobId**.

Input Arguments

`jobId` [integer] Unique number assigned to a job at creation time.

`approver` [string] Account of the person who approved the job. Valid only if approval is required.

CISCO CONFIDENTIAL**cancel**

```
cancel jobId
```

Cancels the job **jobId**.

Input Arguments

jobId [integer] Unique number assigned to a job at creation time.

create

```
create cmd=command [,descr=description] [,owner=user] [,type=string] [, schedule]
```

Creates a job.

Input Arguments

cmd [string] Command required to identify the job.

descr [string] Describes the job.

owner [string] Account of the person who created the job.

type [string] Identifies the job type and subtypes (for example, SWIM:update.)

schedule [string] List of comma-separated fragments that specify when the job will be run initially, how often it is repeated, and the initial schedule state:

- at {date | +minutes}

date specifies the start datetime as a string. Alternatively, +minutes can be used to start the job in *minutes* minutes from the current time.
- repeat {weekly | monthly | daily | month Last Day | monthSameXday | monthLastXday} [(n)]

Schedule the job to run periodically on a calendar basis.
- repeat every n {h | m | s}

Schedule a job to run periodically on a time basis. Using the “every” option, the job will run every *N* hours/minutes/seconds. Using the “after” option, the job will run *N* hours/minutes/seconds after the end of the previous execution.
- repeat after n {h | m | s}

Schedule a job to run periodically on a time basis. Using the “every” option, the job will run every *N* hours/minutes/seconds. Using the “after” option, the job will run *N* hours/minutes/seconds after the end of the previous execution.
- schst= {W | A | R}

Sets the state to Waiting for approval / Approved / Rejected.

delay

```
delay cmd=command [,descr=description] [,owner=user] [,type=string]
```

CISCO CONFIDENTIAL

Creates a job for immediate execution but in the suspended state. The effect is that the job will be run once it is enabled with the **resume** command.

Input Arguments

<code>cmd</code>	[string] Command required to identify the job
<code>descr</code>	[string] Describes the job
<code>owner</code>	[string] Account of the person who created the job
<code>type</code>	[string] Identifies the job type and subtypes (for example, SWIM:update)

delete

```
delete jobId
```

Deletes the job **jobId**.

Input Arguments

<code>jobId</code>	[integer] Unique number assigned to a job at creation time.
--------------------	---

getnextschedule

```
getnextschedule jobId
```

Prints the next scheduled run time for the job when the Instance of Job is scheduled for the future.

Input Arguments

<code>jobId</code>	[integer] Unique number assigned to a job at creation time.
--------------------	---

reject

```
reject jobId rejecter
```

Rejects the job **jobId**.

Input Arguments

<code>jobId</code>	[integer] Unique number assigned to a job at creation time.
<code>rejecter</code>	[string] Account of the person who rejected the job. Valid only if approval is required.

resume

```
resume jobId
```

CISCO CONFIDENTIAL

Resumes the job **jobId** so it can be scheduled.

Input Arguments

`jobId` [integer] Unique number assigned to a job at creation time.

run

```
run cmd=command [,descr=description] [,owner=user] [,type=string]
```

Creates a job and runs it immediately.

Input Arguments

`cmd` [string] Command required to identify the job

`descr` [string] Describes the job

`owner` [string] Account of the person who created the job

`type` [string] Identifies the job type and subtypes (for example, SWIM:update)

schedule

```
schedule jobId schedule
```

Reschedules the job **jobId**.

Input Arguments

`jobId` [integer] Unique number assigned to a job at creation time.

`schedule` [string] See “[cancel](#).”

suspend

```
suspend jobId
```

Suspends the job **jobId** so it will not be scheduled until it is resumed.

Input Arguments

`jobId` [integer] Unique number assigned to a job at creation time.



CISCO CONFIDENTIAL

CHAPTER 19

Using Event Services Software

Event Services Software (ESS) is an asynchronous messaging service that provides a messaging infrastructure based on a publish-and-subscribe paradigm. It enables distributed, loosely coupled interprocess communications.

ESS is one of two event messaging components supplied with CWCS. The other – the Event Distribution System (EDS) – is a means for sending messages from one process to another in a distributed, networked environment. *These two components are disjoint systems and do not work together.*



Note

ESS is the preferred messaging service. EDS (see [Chapter 20, “Using the Event Distribution System”](#)) has been deprecated, and support for it will be withdrawn in a future CWCS release. Cisco urges developers to avoid new development with EDS and begin using ESS as soon as possible.

The following topics explain ESS and how to use it:

- [Understanding ESS Subsystems](#)
- [How Does ESS Work?](#)
- [How Is ESS Organized?](#)
- [Using Tibco’s Rendezvous](#)
- [About Subject Names and Event Formats](#)
- [Using the Lightweight Messaging Service](#)

Understanding ESS Subsystems

The two main subsystems of ESS are Tibco’s Rendezvous and the Lightweight Messaging Service (LWMS). Both contain implementations of the Java Messaging Service (JMS) API.

For information on using Tibco’s Rendezvous, see the “[Using Tibco’s Rendezvous](#)” section on [page 19-2](#). For information on using LWMS, see the “[Using the Lightweight Messaging Service](#)” section on [page 19-8](#).

CISCO CONFIDENTIAL

How Does ESS Work?

Messages are asynchronous requests, reports, or events that contain precisely formatted data that describe specific actions, queries, or declarations. Through the exchange of these messages, applications with distributed processes in a networked environment can coordinate with and track the progress of other applications in the network. Messaging services are synonymous with event services.

Messages are published to subjects. An *event bus* is the logical channel that carries these messages. These messages are sent with a specified message format:

$\text{Subject}_{\text{TIBCO}} = \text{Topic}_{\text{JMS}} = \text{Mailbox}_{\text{LWMS}}$

Subjects are essentially logical message addresses that provide a virtual connection between distributed processes. Subscribers register listeners on subjects and receive asynchronous callbacks as messages arrive. Publish/subscribe messaging is considered loosely coupled because publishers and subscribers may be anonymous and the data to be shared between the distributed processes is formatted into messages that provide a type of buffering mechanism.

A publisher simply pushes a message to a subject name and does not generally know or care who or how many subscribers receive its message. Similarly a subscriber generally does not know what process or processes are sending messages on a given subject. This architecture allows a great deal of flexibility for future changes since it makes few assumptions on who, where, when or how another process will use an event.

Developers need to create unique subject names. In order to decide what their project's namespace should be, you need to know the current recommendation which provides a segregation of subject names so you can independently decide the subject names for your applications without colliding with different application subject names. For recommendations and more information on subject names, see Next Generation Foundation Event Services (ESS) Developer's Guide (ENG-112035).

How Is ESS Organized?

The ESS architecture is divided into two main subsystems:

- A Server Event Bus (SEB) is used between backend processes that perform the actual network management computing functions and is implemented with Tibco Software Inc.'s Rendezvous product.
- A Client Event Bus (CEB) is used for messaging with CiscoWorks web client front ends and is implemented by the internally developed LWMS.

A gateway between the SEB and CEB transparently and automatically forwards messages between the two event buses.

If you need additional information on these ESS subsystems, see Event Services Software (ESS) Primer and Overview in EDCS (ENG-110601).

Using Tibco's Rendezvous

Application developers must use Tibco's Rendezvous product to communicate between backend processes such as your application's processes and other interested processes. The TibcoRV version 7.1 is used for this release of CiscoWorks Common Services.

For more information, see:

CISCO CONFIDENTIAL

- Event Services Software (ESS) Primer and Overview in EDCS (ENG-110601)
- Next Generation Foundation Event Services (ESS) Developer's Guide (ENG-112035), Section 7, sample code for Rendezvous and LWMS
- TB/RV Overview (ENG-110810)

About Subject Names and Event Formats

This section contains the following topics:

- [How Subject Names are Structured](#)
- [Choosing Subject Names and Namespaces](#)
- [Subscribing with Wildcards](#)
- [About ESS Event Formats](#)
- [About Reserved Subject Names](#)

How Subject Names are Structured

This section explains the general structure of Subject names. For specific recommendations, see the “[Choosing Subject Names and Namespaces](#)” section on page 19-3.

Each subject name is a dotted string. For example, *cisco.mgmt.vms.vhm.alarm.voiceGateway*.

The subject namespace forms a hierarchical structure with a prefix portion (the first three dotted elements) that is centrally administered by CANA (Cisco Assigned Numbers Authority). This is mainly to avoid namespace collisions. The remaining part of the subject name is defined by application developers, as shown here:

```
cisco.mgmt.<systemId>.<subsystemId>...<eventClass>.<topic>
cisco.mgmt.<systemId>.<subsystemId>...<eventClass>.<eventSubclass>.<topic>
<--CANA Registered--> <-----organization defined----->
```

The *subsystemId* element identifies the subsystem within the management server.

The *eventClass* element can be used to define a general class of events used within an application. For consistency we will track the event class element names used within EMBU and reuse them whenever possible. For more on Event Classes, see.

The *eventSubclass* element, if present, can be used if there are a large number of events in the event class and further partitioning would be helpful.

The final element *topic* should be used to identify as clearly as possible the type of events that will be published on the complete subject name.

Choosing Subject Names and Namespaces

Follow the subject namespace guidelines shown in [Table 19-1](#) to decide on a subject name that will not cause your application's subject name to collide with other applications.

These guidelines take into account the fact that events may be used for communication in a variety of environments, including between components or services:

- Within one product on the same machine.

CISCO CONFIDENTIAL

- b. Within the same product on different machines.
- c. Within different products on the same
- d. Within different products on different machines.

For example, multiple servers on the same subnet, all with ESS-based network management applications installed on them, can pose a problem for situations like that in (a) above, if broadcast is used instead of multicasting. Unless subject names are chosen carefully, events from one product may be passed to another product on a different server unintentionally. One way to prevent this is to include the IP address of the server in the subject name. If this convention is adopted, each product can subscribe to messages from the services on the same server only.

Table 19-1 Subject Namespace Guidelines

For	Use the Subject Name Prefix
CWCS shared services	cisco.mgmt.cw.<IPAddress_with_underscores>.cmf.<service name>.<event-class>
CWCS per-product services	cisco.mgmt.cw.<IPAddress_with_underscores>.<product-name>.<service name>.<event-class> Where: cw stands for CiscoWorks. <product-name> is the name of the product which uses one instance of a service. In case, different applications in a given product use different instances of a service, then <product-name> should qualify such an application rather than the product itself. An example for <product-name> is RME. If RME applications A and B use two different instances of a service, then <product-name> should be of the form <RME-A> and <RME-B>, to ensure separation among the events between the service as used by application A and the service used by application B. <IPAddress_with_underscores> is the IP address of the host server in the string form "%d.%d.%d.%d" with the dots (".") replaced by underscores ("_"). This can be achieved using the methods in the class com.cisco.nm.cmf.ess.utils.Namespace.
Interproduct communication	cisco.mgmt.cw.<product name> Where <product-name> is the identifier of a product which is the source or publisher of the event.
Communication across multiple product instances (of same or different product types)	cisco.mgmt.cw. <cluster id>.<product name>

When choosing subject names, be aware of the following:

- Generate a dotted name hierarchy for all of your subject names such that related events are grouped. This allows convenient subscription using wildcards.
- Create a unique new subject name for each event group to which subscribers might want to selectively subscribe. For example, if most subscribers are likely to be interested in all events in a given group, put them all under a common subject name. If many or most of the subscribers are likely to be interested only in specific event types then give them each a unique subject name.
- Subject names are case sensitive.
- Limit subject names to less than 100 characters.

CISCO CONFIDENTIAL

- Do not use the reserved subject names listed in the “About Reserved Subject Names” section on page 19-5.

Subscribing with Wildcards

Tibco Rendezvous supports two wildcard characters: the asterisk (*) and greater-than symbol (>). The * character matches one single whole element, but not a partial substring of characters, within an element. The > character matches one or more trailing elements to the right. The semantics of listening to wildcard subjects are shown in Table 19-2.

Table 19-2 Wildcard Subject Semantics¹

Listening to this wildcard name	Matches messages with names like these:	But does not match messages with names like these (reason):
RUN.*	RUN.AWAY RUN.away	RUN.run.run(extra element) Run.away (case differs) RUN (missing element)
Yankees.vs.*	Yankees.vs.Red_Sox Yankees.vs.Orioles	Giants.vs.Yankees (position) Yankees.beat.Sox (beat used instead of vs ²) Yankees.vs (missing element)
.your.	Amaze.your.friends Raise.your.salary Darn.your.socks	your (missing elements) pick.up.your.foot (position)
RUN.>	RUN.DMC RUN.RUN.RUN RUN.SWIM.BIKE.SKATE	SWIN.RUN (position) Run.away (case differs) RUN (missing element)

- This table is taken from the Tibco/RV Concepts Manual.
- LWMS also supports subscriptions with wildcard characters with the single limitation that the publisher must be on the SEB Tibco/Rv side of the event bus.

About ESS Event Formats

ESS supports two types of message body content:

- Text: This can be an unstructured string, or structured ASCII text (such as XML).
- Java Object: The content must implement the java.io.Serializable interface.

About Reserved Subject Names

The following Subject Names are in use and reserved by NMTG:

- subsystemId* Element Identifiers:
 - cisco.mgmt.trx.
 - inventory

CISCO CONFIDENTIAL

- *eventClass* Element Identifiers:
 - .<eventClass>.
 - alarm
 - status
 - test
 - debug
 - command
- Legacy Subject Names:
 - **cisco.mgmt.cmf.**
 - **cisco.mgmt.cmf.events.ltGateway.control**
 - cisco.mgmt.cmf.events.ltGateway.allSubjects
 - **cisco.mgmt.vhm.**
 - **cisco.mgmt.vhm.alarm.summary**
 - cisco.mgmt.vhm.alarm.vhmserver
 - cisco.mgmt.vhm.alarm.dfmserver
 - cisco.mgmt.vhm.alarm.status
 - cisco.mgmt.vhm.alarm.gershwin
 - **cisco.mgmt.vhm.invupdate.summary.add**
 - cisco.mgmt.vhm.invupdate.summary.delete
 - cisco.mgmt.vhm.invupdate.summary.update
 - **cisco.mgmt.vhm.invupdate.status.add**
 - cisco.mgmt.vhm.invupdate.status.delete
 - cisco.vpnsc
 - cisco.cns

Support for Map Messages

From Common Services 3.0 Service Pack 1, map messages are supported. Publish and subscribe can happen via map messages. A map message is a message whose body contains a set of name-value pairs where names are Strings and values are Java primitive types. This map message feature is not supported for LWMS and JMS.

Sample code for Map message Publisher:

```
import com.cisco.nm.cmf.jms.*;
import java.io.*;
import javax.jms.*;

public class EssJtibMapPub{
    private static String subject = "cisco.mgmt.cw.cmf.jrm";
    private TopicConnectionFactory Factory;
    private TopicConnection conxn;
    private Topic topic;
    private TopicSession session;
    private TopicPublisher publisher;
```

CISCO CONFIDENTIAL

```

private MapMessage msg;

public static void main(String args[]){

    new EssJtibMapPub().start();
}
private void start(){

    try{
        Factory = new TopicConnectionFactoryImp();
        conxn = Factory.createTopicConnection();
        session = conxn.createTopicSession(false, javax.jms.Session.AUTO_ACKNOWLEDGE);

        topic = session.createTopic(subject);
        publisher = session.createPublisher(topic);

        msg =session.createMapMessage();
        msg.setString("map", "I am a Tibco MAP message !!!!!.");
        msg.setJMSReplyTo(new TopicImp("mapreply"));
        msg.setJMSDestination(topic);
        msg.setJMSPriority(7);
        publisher.publish(msg);
    }
    catch(Exception ex){ ex.printStackTrace();};
}
}

```

Sample code for Map message Subscriber:

```

import java.io.*;
public class EssJtibMapSub implements javax.jms.MessageListener {

    private static String subject = "cisco.mgmt.cw.cmf.jrm";
    private static boolean isFileBased = true;
    private javax.jms.TopicConnectionFactory Factory;
    private javax.jms.TopicConnection conxn;
    private javax.jms.Topic topic;
    private javax.jms.TopicSession session;
    private javax.jms.TopicSubscriber subscriber;

    public static void main(String args[]){
        new EssJtibMapSub().start();
    }

    private void start(){

        try{

            Factory = new com.cisco.nm.cmf.jms.TopicConnectionFactoryImp();
            conxn = Factory.createTopicConnection();
            session = conxn.createTopicSession(false, javax.jms.Session.AUTO_ACKNOWLEDGE);

            topic = session.createTopic(subject);
            subscriber= session.createSubscriber(topic);
            subscriber.setMessageListener(this);
            conxn.start();
        }catch(Exception ex) { ex.printStackTrace();};
    }

    public void onMessage(javax.jms.Message msg){
        javax.jms.MapMessage mapmsg;
        try{
            if (!(msg instanceof javax.jms.MapMessage) ) return;

```

CISCO CONFIDENTIAL

```

        mapmsg = (javax.jms.MapMessage)msg;
String map_msg = mapmsg.getString("map");
return;
    }
    catch( Exception ex ) { ex.printStackTrace(); }
}

```

Using the Lightweight Messaging Service

The Lightweight Messaging Service (LWMS) is a light weight XML-based messaging service used primarily between client desktops and the Common Services server. The client desktops are authenticated before sending messages.

The LWMS service provides:

- XML message format (without requiring an XML parser on the client)
- HTTP transport (for traversing firewalls)
- A zero-administration client—No administration configuration is required to create mailboxes or send or receive messages
- Native API and Java Messaging Service (JMS) API support
- A small client footprint:
 - Less than 50 KB (using native LWMS API)
 - Less than 90 KB (using JMS API)
- Message send queues, with high and normal priorities
- Efficient message filtering (using JMS message selectors)
- Support for publish/subscribe and point-to-point messaging models:
 - Publish/subscribe messaging services are a type of distributed interprocess communication (IPC) generally intended for loosely coupled asynchronous communications between software processes. Publish/subscribe messaging is useful for one-to-many, many-to-one, and many-to-many communication relationships.
 - Point-to-point (P2P) messaging is a special case of publish/subscribe messaging where there is a single publisher and a single subscriber.

The following topics describe the Lightweight Messaging Service and how to use it in your applications:

- [Understanding LWMS](#)
- [Configuring LWMS](#)
- [Using the LWMS API](#)
- [Using the JMS API](#)
- [LWMS Command Reference](#)

For more information, see:

- *CMF Lightweight Messaging Service (LWMS) Functional Specification*, ENG-58367
- *CMF Lightweight Messaging Service (LWMS) Overview Presentation*, ENG-72595
- JMS (Java Messaging Service) 1.0.2 API Specification at Sun's web site

CISCO CONFIDENTIAL

Understanding LWMS

The following topics describe the components and features of the Lightweight Messaging Service (LWMS):

- [About the LWMS Components](#)
- [How LWMS Works](#)
- [About LWMS Message Queues](#)
- [About JMS API Support](#)
- [About LWMS Server Logging](#)
- [About LWMS Usage Assumptions](#)

About the LWMS Components

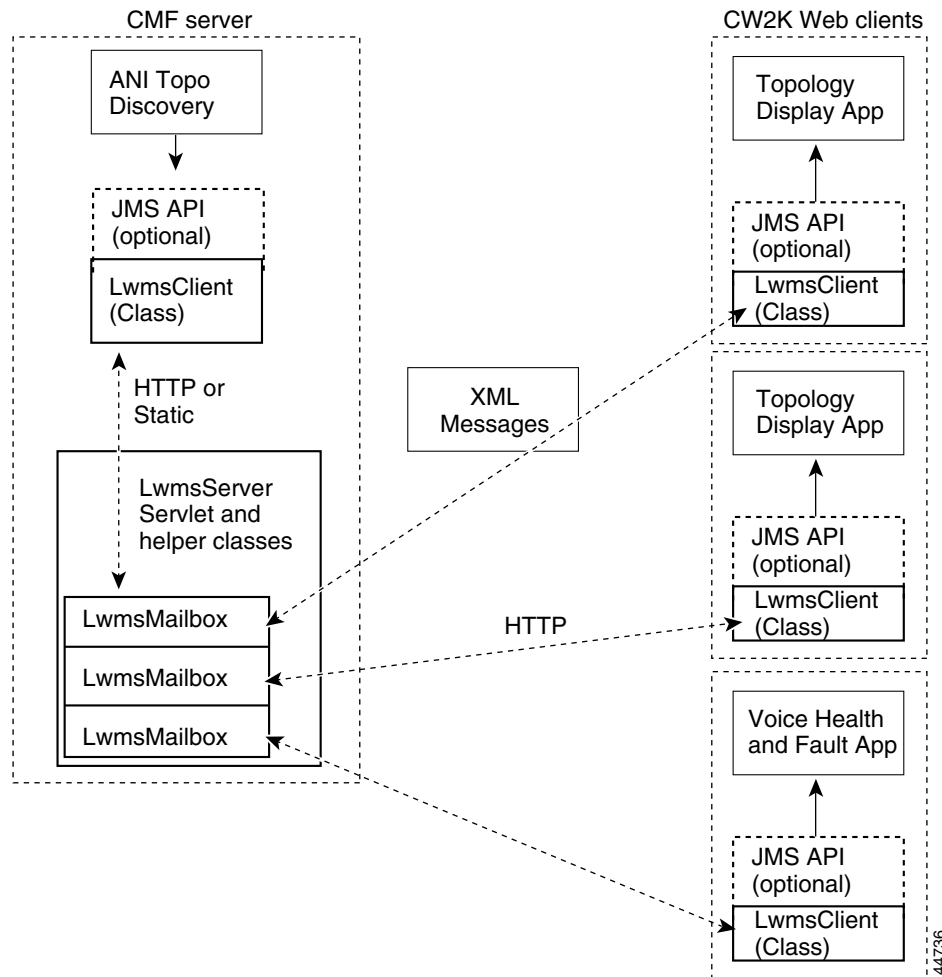
LWMS consists of two main components:

- **LwmsClient:** Provides the client interface for creating mailboxes and sending and receiving messages. The Java class that implements this interface can be used by applets in the desktop client web browser environment to send and receive messages with an LwmsServer.
- **LwmsServer:** Runs as part of the CWCS server infrastructure and handles XML message distribution.

Two more components provide additional services:

- **LwmsMailbox:** Stores and manages messages and message forwarding.
- **LwmsMessage:** Simplifies and controls creation of XML messages.

[Figure 19-1](#) shows how all of these components work together.

CISCO CONFIDENTIAL**Figure 19-1** LWMS Usage Example**How LWMS Works**

LWMS supports three primary operations:

1. Creating mailboxes
2. Publishing messages to mailboxes
3. Listening for new messages to mailboxes.

An LWMS mailbox may have one or more senders and one or more receivers; internally, LWMS makes no distinctions between these usage types. A named mailbox simply receives messages sent to it, time stamps and ages them according to the time-to-live (TTL) field in each message, and forwards copies of new messages to each polling client.

Each LwmsClient keeps track of the last message it has seen for a given mailbox. On subsequent polls only messages later than this are forwarded to the client.

CISCO CONFIDENTIAL

About LwmsClient

To provide the client interface for creating mailboxes and sending and receiving messages, the LwmsClient:

- Makes a single URL connection to the CWCS Server.
- Uses HTTP to send messages to the LwmsServer.
- Uses prioritized message queues (see the [“About LWMS Message Queues”](#) section on page 19-12).
- Performs the following for each mailbox with a registered listener:
 - Uses HTTP to poll the LwmsServer every N seconds to request new messages.
 - Remembers the latest message time stamp that it has seen and requests new messages from that point in the message stream.
 - Forwards new messages to registered listeners.
 - Supports non-blocking, asynchronous listener registration (callbacks).
- Allows the user to:
 - Set the maximum send-queue size.
 - Set the receive polling interval (the default is 1.5 seconds; the configurable range is from 0.5 to 30 seconds).
 - Set the send push interval for normal-priority messages (the default is 1 second; the configurable range is from 0.5 to 30 seconds).
 - Set the maximum number of messages returned by a server per request.

About LwmsServer

To handle message distribution, the LwmsServer:

- Maintains a set of named mailboxes that store messages until they expire.
- Timestamps new messages and adds them to the mailbox.
- Returns any new messages to clients when polled.

About LwmsMailbox

The LwmsMailbox component stores and manages messages and handles message forwarding. An LwmsMailbox:

- Receives the messages sent to it.
- Ages messages according to the value in the TTL (time-to-live) field in the message header.
- Forwards copies of new messages to each polling client

Each LwmsClient keeps track of the latest time stamp it has seen for a given mailbox. On subsequent polls, only messages later than this time stamp are forwarded to its client.

- Supports filtering based on message header fields

An LwmsMailbox may have one or more senders and one or more receivers.

About LwmsMessage

The LwmsMessage component simplifies and controls the creation of XML messages. The body of an LwmsMessage may be either:

CISCO CONFIDENTIAL

- A string. The string may be structured (normally, XML) or unstructured.
- An object. The object must be Base64-encoded on wire, and the receiver can cast back to the original object type.

LwmsMessage supports extensible message header elements via the `msg.addHeaderElement()` class. For example:

```
msg.addHeaderElement("deviceType", "Cat6000");
```

In this example, the client application could perform filtering operations based on the contents of the message header.

For more information about LWMS message formats, see *CMF Lightweight Messaging Service (LWMS) Functional Specification*, ENG-58367.

About LWMS Message Queues

LWMS supports two priority levels for sending messages:

- **NORMAL:** LWMS bundles and sends messages with normal priority periodically, at a predefined push interval (for example, once every second). Normal-priority messages are always received in the order they are sent, are the most efficient to transmit, and provide the best throughput for large bursts of messages.
- **HIGH:** LWMS sends high-priority messages to the LwmsServer message server immediately. This priority level provides the lowest message latency but has a higher overhead, so use it only when necessary.

By default there is only one high priority push thread per LWMS client, which preserves the message sending order. If, however, there is more than one thread, high-priority messages may be received out of order. Multiple high priority pusher threads are recommended when higher throughput performance is required (see the [“Configuring Client Properties”](#) section on page 19-13).


Note

A maximum rate limit is enforced when sending high-priority messages. If this rate is exceeded, the sending thread is blocked until the high-priority send queue and send thread-pool load drops to a lower threshold.

About JMS API Support

LWMS supports messaging via the Java Messaging Service API, Version 1.0.2. For a list of the subset of JMS services that LWMS supports, see the [“JMS to LWMS Mappings”](#) section on page 19-21.

About LWMS Server Logging

The LWMS Server outputs exceptions and important status information to the file `lwms.log` in the CWCS log directory.

About LWMS Usage Assumptions

LWMS was designed to meet the following usage limits:

- Typical throughput: less than 100 messages/second
- Typical fan in (number of publishers): less than 10

CISCO CONFIDENTIAL

- Typical fan out (number of subscribers): less than 25

About Tibco-LWMS Gateway Support

LWMS implements the Client Event Bus (CEB) in the CWCS event architecture. There is also a Server Event Bus (SEB) which is currently implemented by Tibco's Rendezvous product.

When messages need to transit on both the CEB and the SEB buses, a gateway is required between LWMS and Tibco. The gateway handles message traffic in both directions:

- Gateway publishing to LWMS (Tibco to LWMS)

LWMS provides control messages on a specified topic that notify the gateway about the topics in the LWMS bus that have active listeners. The gateway forwards messages published on these topics on the Tibco bus to LWMS.

Control messages are sent to the gateway when:

- A listener is added to a topic that was not in the previous control message.
- A topic no longer has any registered listeners (as determined by specific listener deregistration or listener table aging).

- Gateway subscribing to LWMS (Tibco from LWMS)

The gateway subscribes to a special topic that will forward all messages published in the LWMS domain. This special topic is designed to prevent messages that were published earlier by the gateway from echoing back to the gateway. This is similar to the JMS NoLocal option.

Configuring LWMS

Two configuration files are loaded at startup to provide operating properties to LWMS. The following topics describe how to modify these files to configure your LWMS implementation:

- [Configuring Client Properties](#)
- [Configuring Server Properties](#)

Configuring Client Properties

To configure the LWMS client properties, use an ASCII editor to edit the `LwmsClientProperties` file.

Runtime Location	<code>\$NMSROOT/lib/classpath/com/cisco/nm/cmflwms/LwmsClientProperties.xml</code> where <code>NMSROOT</code> is the directory in which the product is installed. This directory also contains the supporting files.
-------------------------	---

The client properties file uses the tags shown in [Table 19-3](#).

CISCO CONFIDENTIAL**Table 19-3** *LwmsClientProperties.xml File Tags*

Tag	Attributes	Default	Description
serverPollInterval	VALUE	1500	Server mailbox polling interval, in milliseconds.
normalMsgPushInterval	VALUE	1000	Pushes normal messages to server, in milliseconds.
numHighPriorityPusherThreads	VALUE	1	Set to 1 to preserve message order. Higher values may increase throughput.
maxHighPriorityMsgQueueSize	VALUE	500	Blocks next sender if queue is full.
useGZIPioStreams	VALUE	False	Compresses applet-servlet streams.

Example 19-1 shows a typical LWMS client configuration file.

Example 19-1 *LWMS Client Configuration File*

```
<?xml version="1.0"?>
<lwmsClientProperties>
  <serverPollInterval VALUE="1500" /> # server mailbox polling interval in ms
  <normalMsgPushInterval VALUE="1000" /> # push normal messages to server, in ms
  <numHighPriorityPusherThreads VALUE="1" /> # set to 1 to preserve message order.
    # Higher values may increase throughput
  <maxHighPriorityMsgQueueSize VALUE="500" /> # next sender is blocked if queue is full
  <useGZIPioStreams VALUE="false" /> # compress applet-servlet streams
</lwmsClientProperties>
```

**Note**

Changes to client properties take effect on the next applet startup or reload in the client browser.

Configuring Server Properties

To configure the LWMS Server properties, use an ASCII editor to edit the `LwmsServerProperties.xml` file.

Runtime Location `$NMSROOT/lib/classpath/com/cisco/nm/cm/lwms/LwmsServerProperties.xml`
 where `NMSROOT` is the directory in which the product is installed. This directory also contains the supporting files.

The server properties file uses the tags shown in Table 19-4.

Table 19-4 *LwmsServerProperties.xml File Tags*

Tag	Attributes	Default	Description
msgAgeingInterval	VALUE	60	Message aging interval, in seconds.
tempMailboxAgeingInterval	VALUE	30	Temporary mailbox aging interval, in seconds. Also used for the Tibco-LWMS Gateway.

CISCO CONFIDENTIAL**Table 19-4** *LwmsServerProperties.xml* File Tags

Tag	Attributes	Default	Description
maxMsgsPerPoll	VALUE	500	This value divided by client poll interval determines maximum throughput.
autoCreateMailboxes	NAME	(none)	Mailboxes created at LWMS Server startup time. If TTL (time-to-live)="0" then the mailbox is persistent (useful for JMS).

Example 19-2 shows a typical LWMS Server configuration file.

Example 19-2 *LWMS Server Configuration File*

```
<?xml version="1.0"?>
<lwmsServerProperties>
  <msgAgeingInterval VALUE="60" />
  <tempMailboxAgeingInterval VALUE="30" />
  <maxMsgsPerPoll VALUE="500" />
  <autoCreateMailboxes>
    <create NAME="cisco.mgmt.ani.event" TTL="0" />
    <create NAME="JmsTestTopic" TTL="0" />
  </autoCreateMailboxes>
</lwmsServerProperties>
```

**Note**

Server properties are read in on the first access to the LWMS servlet after the servlet engine starts. Therefore, after changing any server properties, you must restart the servlet engine.

Using the LWMS API

The following topics describe how to use the native LWMS API to perform typical messaging tasks:

- [Creating a Mailbox with LWMS](#)
- [Posting a Message to a Mailbox with LWMS](#)
- [Polling Mailboxes for New Messages with LWMS](#)
- [Removing a Message Listener with LWMS](#)
- [Filtering Messages with LWMS](#)

Creating a Mailbox with LWMS

Use the following code to get a reference to an `LwmsClient` singleton object and create a mailbox:

```
import com.cisco.nm.cmf.lwms.*;
LwmsClient lc= LwmsClient.getInstance( "http://server:1741", cookie_value, user, is_cli );
lc.createMailbox("mbox");
```

CISCO CONFIDENTIAL**Posting a Message to a Mailbox with LWMS**

To create a message in LWMS, set the message properties and send the message:

```
LwmsMessage msg = new LwmsMessage();
    msg.setToMailbox ("mbox");
    msg.setMsgBody("Hello Gang");
lc.sendMessage(msg);
```

To get a reply, add these lines:

- Create a replyMailbox:


```
lc.createMailbox("Replymbox");
```
- Set the replyTo field in the message:


```
msg.setReplyTo("Replymbox");
```
- Listen on that mailbox:


```
lc.addMsgListener("Replymbox", this);
```

Polling Mailboxes for New Messages with LWMS

To register a message listener with LWMS:

```
lc.addMsgListener("mbox", this);
```

where the current object:

- Implements the LwmsMsgListener interface
- Has a public newMessage(LwmsEvent) method

Removing a Message Listener with LWMS

To unregister a message listener with LWMS:

```
void removeMsgListener(String mailbox, LwmsMsgListener msgListener);
```

Filtering Messages with LWMS

An LWMS message consumer may optionally specify a message filter for each mailbox at listener registration time. This filter has the form of a Boolean expression containing equality, relational, or logical operators. Each message that evaluates to true based on the filter criteria will be returned to the listener. Filters check only the standard and extended headers.

[Table 19-5](#) summarizes the filter operator types LWMS supports.

Table 19-5 *LWMS-Supported Filter Operators*

Operator	Types
Equality	=, <> (equal to, not equal to)
Relational	>, <, >=, <= (greater than, less than, greater than or equal to, less than or equal to)
Logical	AND, OR

CISCO CONFIDENTIAL

Base Filter Expression (bfe) Syntax

The base filter expression uses this format:

```
match_string operator literal_value
```

where:

- The white space before and after the *operator* field is required.
- The *match_string* field must begin with a ‘<’ and end with either:
 - A ‘>’ for a standard XML element header field
 - An ‘=’ for an attribute in an XML empty element header field
- The *match_string* field identifies a message header field and is of one of the following two forms:
 - <tag>—identifies a standard XML element value
 - <tag VALUE=—identifies an XML element attribute
- If *match_string* ends in “>”, then its value is a substring extraction up to the next “<” character.

For example, a subscriber with

```
filter = "<foo> = 'bar'"
```

would receive messages with an extended header <foo>bar</foo>.

- If *match_string* ends in “=”, then its value is a substring extraction from the first quote mark up to the next quote mark.

For example, a subscriber with

```
filter = "<foo VALUE= < 7"
```

would receive messages with an extended header of <foo VALUE="5"/> (or any value less than 7).

- The *operator* is one of: =, <>, >, <, >=, <=.
- String and character literals are enclosed in single quotes (for example, 'stringValue').
- Each base filter expression evaluates to Boolean true or false.

Filter Strings Syntax

Filter strings supplied to the listener registration call use this syntax:

```
String filter= "bfe [AND bfe | OR bfe]";
```

where:

- Additional base filter expression terms enclosed in brackets are optional.
- Additional base filter expression terms must be preceded by either AND or OR.

Evaluation Rules

LWMS evaluates filter rules as follows:

- Base filter expression evaluation order is left to right. Parentheses are not supported.
- If the LwmsServer cannot parse the filter, it is ignored and all new messages on the mailbox are returned.

Sample Filters

Here are two separate examples of typical filters:

```
String filter_1= "<from> = 'AniServer'";
```

CISCO CONFIDENTIAL

```
String filter_2= "<deviceType> = 'Cat6000' AND <deviceResets> > 5";
```

Using the JMS API

The following topics describe how to use the JMS APIs to perform typical messaging tasks:

- [Creating a Mailbox with JMS APIs](#)
- [Posting a Message to a Mailbox with JMS](#)
- [Polling Mailboxes for New Messages with JMS](#)
- [Removing a Message Listener with JMS](#)
- [Using JMS Message Selectors](#)

Creating a Mailbox with JMS APIs

To start a connection and a session and get a topic publisher using the JMS API:

```
import com.cisco.nm.cmf.lwms.jms.*;
import javax.jms.*;
TopicConnectionFactory tcf=
    new TopicConnectionFactoryImp (protocol, hostName, port);
TopicConnection tc= tcf.createTopicConnection();
TopicSession sess=
    tc.createTopicSession (false, DUPS_OK_ACKNOWLEDGE);
Topic pubTopic= new lwms.jms.TopicImp ("JmsTestTopic");
TopicPublisher publisher= pubSession.createPublisher (pubTopic);
tc.start();
```

Posting a Message to a Mailbox with JMS

To post a message to a topic using the JMS API:

```
javax.jms.TextMessage message= sess.createTextMessage();
message.setText ("Hello from LWMS/JMS");
message.setJMSDestination (pubTopic);
publisher.publish (message);
```

Polling Mailboxes for New Messages with JMS

To subscribe to a topic using the JMS API:

```
import com.cisco.nm.cmf.lwms.jms.*;
import javax.jms.*;

TopicConnectionFactory tcf=
    new TopicConnectionFactoryImp (protocol, hostName, port);
TopicConnection tc= tcf.createTopicConnection();
TopicSession sess= tc.createTopicSession (false, DUPS_OK_ACKNOWLEDGE);
Topic subTopic= new lwms.jms.Topic ("JmsTestTopic");
TopicSubscriber subscriber= sess.createSubscriber (subTopic);
subscriber.setMessageListener (this);
tc.start();
```

Removing a Message Listener with JMS

To unregister a JMS message listener:

CISCO CONFIDENTIAL

```
TopicSubscriber.setMessageListener(null)
```

Using JMS Message Selectors

LWMS supports JMS message selectors. The JMS message-selector syntax is identical to the LWMS message filter syntax (see the “[Filtering Messages with LWMS](#)” section on page 19-16).

LWMS Command Reference

These topics provide reference information about LWMS and JMS API calls:

- [LWMS Native API Messaging Methods](#)
- [JMS to LWMS Mappings](#)

LWMS Native API Messaging Methods

The following tables list the public Java interfaces needed to create a mailbox, send a message, and perform other messaging tasks.

Table 19-6 *LwmsClient Public Methods*

Returns	Syntax and Description
Setup	
Boolean	checkMailboxExists (String <i>mailbox</i>); Returns true if named mailbox already exists on LwmsServer.
Boolean	createMailbox (String <i>mailbox</i>); Creates a mailbox on server.
Boolean	createMailbox (String <i>mailbox</i> , int <i>ttl</i>); Creates a temporary mailbox on server, will be deleted after TTL (time-to-live) seconds. For a permanent mailbox, set TTL=0. Returns success.
Sender (Message Producer)	
void	sendMsg (LwmsMessage <i>msg</i>); Sends a message to server. Note: Message object should not be mutated after calling.
Receiver (Message Consumer)	
void	addMsgListener (String <i>mailbox</i> , LwmsMsgListener <i>listener</i>); Registers a message listener.
void	addMsgListener (String <i>mailbox</i> , LwmsMsgListener <i>listener</i> , String <i>filter</i>); Registers a message listener with a filter specification.
void	removeMsgListener (String <i>mailbox</i> , LwmsMsgListener <i>msgListener</i>)

CISCO CONFIDENTIAL**Table 19-7 LwmsMessage Message Creation Methods**

Returns	Syntax and Description
void	addHeaderElement (String name, String value); Adds “<name>value</name>”.
void	setFrom (String from);
void	setPriority (int priority); Allowed values: <ul style="list-style-type: none"> • LwmsMessageIF.HIGH • LwmsMessageIF.NORMAL
void	setMsgBody (String msgBody);
void	setMsgBody (java.io.Serializable msgBody); Object will be serialized and Base64 encoded.
void	setMsgName (String msgName);
void	setMsgType (String msgType);
void	setReplyTo (String replyToMailbox);
void	setSenderTimeStamp (long senderTimeStamp);
void	setTimeToLive (long timeToLive); Messages TTL in seconds.
void	setToMailbox (String toMailbox);

Table 19-8 LwmsMsgListener Interface

Returns	Syntax and Description
void	newMessage (LwmsMsgEvent event); Callback method in listener.

Table 19-9 LwmsMsgEvent Methods

Returns	Syntax and Description
String[]	getExtendedHeaderElementNames ();
String	getExtendedHeaderElementValue (String name);
String	getFrom ();
long	getMboxTimeStamp ();
String	getMsg (); Returns whole XML message.
Object	getMsgBody (); Returns message body: <ul style="list-style-type: none"> • If MsgBodyType== STRING then cast as String. • If MsgBodyType== OBJECT then cast as appropriate for your application.
String	getMsgBodyType ();
String	getMsgID ();
String	getMsgName ();
String	getMsgType ();

CISCO CONFIDENTIAL**Table 19-9** *LwmsMsgEvent Methods (continued)*

Returns	Syntax and Description
int	<code>getPriority()</code> ;
String	<code>getReplyToMailbox()</code> ;
long	<code>getSenderTimeStamp()</code>
long	<code>getTimeToLive()</code> ;
String	<code>getToMailbox()</code> ;

JMS to LWMS Mappings

The following tables list significant mappings between the JMS and LWMS APIs and message fields.

For a complete list of JMS to LWMS mappings, see the *CMF Lightweight Messaging Service (LWMS) Functional Specification*, ENG-58367.

Table 19-10 *JMS to LWMS API Mappings*

JMS	LWMS
<pre>publisher.publish(msg); publisher.publish(topic, msg); publisher.publish(msg, deliveryMode, priority, ttl); publisher.publish(topic, msg, deliveryMode, priority, ttl);</pre>	<p>Converts JMS Message to a LWMS Message per the mappings in Table 19-11 and then calls <code>LwmsClient.sendMessage()</code>.</p> <p>If <code>publish()</code> is called with TTL (time-to-live), the TTL should be set to at least two times the client polling interval.</p> <p>If <code>ttl=0</code> (in JMS, this means the message never expires), LWMS sets the message's TTL to one hour; this is done because LWMS architecture specifies that messages must expire eventually.</p>

Table 19-11 *JMS to LWMS Message Field Mappings*

JMS	LWMS
Body	body
JMSCorrelationID	If present, this field is mapped to an extended header field named <code>JMSCorrelationID</code> .
JMSDeliveryMode	LWMS supports the equivalent of the JMS <code>NON_PERSISTENT</code> delivery mode
JMSDestination:Topic (topic name string is used)	<code>toMailbox</code>
JMSExpiration	Mapped to an extended header field of the same name. Calculated as the sum of the TTL (time-to-live) value in the <code>publish()</code> method call and the client's current time.
JMS Message Properties	LWMS Message Extended Header fields
JMS Message types: TextMessage or ObjectMessage	<code>msgBodyType</code> (<code>TYPE="STRING"</code> or <code>TYPE="OBJECT"</code>)
JMSMessageID	LWMS <code>msgID</code>

CISCO CONFIDENTIAL**Table 19-11 JMS to LWMS Message Field Mappings (continued)**

JMS	LWMS
JMSPriority	Priority
LWMS → JMS	JMS → LWMS
Normal → 2	0-4 → Normal
High → 7	5-9 → High
JMSRedelivered	not mapped
JMSReplyTo:Topic (topic name string is used)	replyToMailbox
JMSTimeStamp	senderTimeStamp The LWMS-JMS client generates this time stamp in the client VM when the publisher.publish() method is called.
JMSType	If present, this field is mapped to an extended header field named JMSType.
Time-to-live (TTL) value from publish() method call, or default value of 60 seconds if not specified	timeToLive
no direct JMS equivalent	from
no direct JMS equivalent	msgName
no direct JMS equivalent	mboxTimeStamp



CISCO CONFIDENTIAL

CHAPTER 20

Using the Event Distribution System

The Event Distribution System (EDS) was the event-distribution software supplied with CMF, the predecessor system of CWCS. For backward compatibility, it is also supplied with this release of CWCS. EDS provides a means for sending messages from one process to another in a distributed, networked environment.

EDS is one of two event messaging components supplied in this release of CWCS. The other – Event Services Software (ESS) – is an asynchronous messaging service providing a publish-and-subscribe infrastructure and allowing distributed, loosely coupled interprocess communications. *These two components are disjoint systems and do not work together.*



Note

EDS has been deprecated in favor of ESS (see [Chapter 19, “Using Event Services Software”](#)). CWCS currently supports EDS for applications that are still using it, but this support will be withdrawn in a future release. Cisco urges developers to avoid new development with EDS and begin using ESS as soon as possible.

The following topics describe EDS and how to use it:

- [About the EDS Components](#)
- [Using the EDS Programmatic Interface](#)
- [Using EDS to Publish Events](#)

For more information about EDS, see the *EDS v1.1 System Functional Specification*, EDCS ENG-25706.

About the EDS Components

The following topics discuss each of the EDS components:

- [About the EDS Event Server](#)
- [About the EDS Event Message](#)
- [About the EDS Atom Service](#)
- [About the EDS Manager](#)
- [About the EDS Class Loader](#)
- [About the EDS New Event Message Fields](#)
- [About the EDS Event Logger](#)
- [About the EDS Event Logger Display](#)

CISCO CONFIDENTIAL

- [About the EDS Named Event Filter Service](#)
- [About the EDS Event to Trap Converter](#)
- [About the EDS Trap to Event Converter](#)

About the EDS Event Server

The Event Distribution System (EDS) is used to distribute event messages from the creator of the events, the Event Source (ES), to the recipient of the events, the Event Consumer (EC). The event is a data packet or message that contains pertinent information associated with an incident in the things being monitored or managed. Each EC registers an Event Profile (EP) or filter, with the EDS that describes the events in which the EC is interested.

The EC defines a Java boolean function that is used to evaluate the events as they come in. If the boolean function returns true, the event is passed to the EC.

About the EDS Event Message

The Event Message is implemented as a Java class. For C++ event sources, the event message is an IDL data structure. This IDL data structure is sent to EDS. The event sources can provide a Java class that can read the IDL data structure and create an instance of the Event Message Java class. For Java event sources, they can create the Event Message Java class and send that directly. After the event Java class is initialized, it is serialized and sent through the EDS. It is de-serialized in the EDS and sent through the event filter. If the event passes the filter, the serialized event is sent to the interested EC. The Java class is de-serialized in the EC and then the event is processed in the EC. Not only is syntactical information sent with the event message, such as event creation time and event ID, but semantical information can also be sent, such as methods to display user readable messages or conversion to useful traps.

About the EDS Atom Service

The EDS Atom Service maintains the definition of the atoms used to define the elements of the events. Atoms are packages of information that associate an integer value atom number with a string atom name, description, and a locale dependent message. An atom may also have a hierarchical relationship with other atoms. The main purpose of an atom is to let a small piece of data (integer) flow in the event, instead of a much larger stream of data such as a text string. It also allows for internationalization of these text strings.

The Atom Service keeps the definition of each atom as well as a locale-correct message that can be displayed about the atom. It can be dynamically updated so that the system does not have to be stopped and started to define new atoms that appear in the event message.

About the EDS Manager

The EDS Manager keeps track of all the CORBA objects that are part of the EDS system. It allows the EDS system to be administered.

CISCO CONFIDENTIAL

About the EDS Class Loader

Since the filters provided by the event consumers and the event messages themselves are instances of Java classes, a class loader is needed to ensure that EDS and the event consumers can get access to the Java class files.

About the EDS New Event Message Fields

New fields added to event messages are delivery type, time to live and chain pointer.

About the EDS Event Logger

The Event Logger (EL) maintains the current event messages that flow through EDS. This component provides search capability on these events. Searching and grouping of historical events aids in problem determination.

About the EDS Event Logger Display

The Event Logger Display (ELD) is used to display events received from EDS as well as events from the event logger.

About the EDS Named Event Filter Service

The Named Event Filter Service provides the ability for the user to define event filters that can be used by other components of the network management system. These event filters are given a name. The Event Logger and Event to Trap facility will use the Named Event Filter Service.

About the EDS Event to Trap Converter

The Event to Trap Converter (ETC) is used to send traps to an NMS such as HP OpenView. After the events are received, the event to trap converter calls the toTrap() method of each event class. It then constructs a trap from the returned information of that method, and then sends it to the configured NMS.

About the EDS Trap to Event Converter

The EDS Trap to Event component is used to convert incoming traps to events that can be used by other event consumers.

CISCO CONFIDENTIAL

Using the EDS Programmatic Interface

Each EDS component has its own API. The APIs for the EDS are defined as Java classes for both event sources and event consumers. There are C++ APIs for event sources and event consumers. The Event Logger Display provides Java constructors so that it can be instantiated from within another Java application. All the other APIs are in the CORBA IDL.

The following topics discuss the EDS interface components:

- [About EDS Events](#)
- [Formatting EDS Events](#)
- [Defining and Registering EDS Event Atoms](#)
- [Using the EDS Atom Definition File](#)
- [Using the Atom Service Executables](#)
- [Using the EDS Java Interface Classes](#)
- [Registering EDS Application Events](#)
- [Using the EDS Trap to Event Service](#)
- [Using the EDS Trap Receiver Framework](#)
- [Using the Generic Consumer Framework](#)

About EDS Events

EDS events must have a required set of attributes to permit generic processing of the event data. Allowing event creators to provide the data in any format they choose makes it harder for users to specify the events they want to view. For example, if the user wants to see all the events pertaining to a router named “MainRouter.enterprise.com”, it will be difficult to filter on or search for them if one event creator calls the attribute that contains the router name “DeviceName”, while another chooses “Device” and yet another chooses “resource”. The same can be said for event categories, such as security events. If we have only a huge list of all events, the user will have to iterate through that list to find all the security events, and are likely to miss at least some of the security events in which they are interested.

In the CWCS event model, the resource each event concerns is defined in a resource list. Each resource in the list has a type and a unique name. Each resource in the list provides details defining what the event is about. Each event must have a resource list and have at least three resources defined: the device, the device component, and the device resource. The device must also contain the IP address of the device. These fields should be populated. If these fields are not filled in, it will be impossible to search on these events in the event logger, and the user will not be able to tell what object the event refers to when shown in the event log display. If it is not possible to set the resource list to anything, it can be set to null.

Most of the data in the event can be defined as integers, with mappings between each integer and its meaning. An atom registry is required to define these integer mappings. The integers need to be defined company-wide and assigned by an “event police” group. The event atom registry allows retrieval of display messages given an integer mapping, and also provides updates to currently running systems.

Some event situations may not permit you to populate all the required fields. In these situations, set the fields to an empty value, such as 0 for an atom value or a string with a value of “”. Do not use a null string; EDS uses CORBA for interprocess communications and IIOP does not permit sending null strings. Be aware that your users will be able to see the contents of all events, and will set filters and

CISCO CONFIDENTIAL

perform searches on those events. The better the required fields are filled in, the easier it is for users to understand what is happening in the network, since they will be applying their filters or queries to a more complete group of events.

Most events will have additional data that describe what the event is about. This data is defined as attributes in the Java class that extends the event base class and as an IDL data structure. This data should be described using metadata.

Formatting EDS Events

The required event fields are listed here. A detailed explanation of each field follows the list.

- Event Category: Category of this event
- Event ID: Unique identifier of the event in this category
- Event Severity: Severity of this event
- Resource List: List of resources
- Time Stamps: GMT time the event was created and sent
- Time to Live: Time to live for this event
- Delivery Type: Delivery type for this event
- Chain Pointer: Unique identifier of related event
- Event UID: Unique identifier of the instance of this event
- Object ID: CORBA object reference of the service that is responsible for managing what this event is about
- Application UID: Unique identifier of the instance of the application sending this event
- Meta-data: Generic definition of the unique data included with this event.
- URL List: List of URLs to provide additional information about this event

Defining and Registering EDS Event Atoms

CWCS needs a service to give definition to atoms. For example, event categories, event IDs, resource types and address types (not a complete list) can all be defined using these atoms. Then, by using the atoms in the event, the size of the event can be minimized. Applications must have a way to define these atoms and update them dynamically when the application is running.

Atom definitions use the following defined hierarchy to allow them to be grouped (the location in the event where this value should appear is given in parenthesis):

- Event Atoms
- Event Category
- Threshold (eventCategory)
 - event IDs*
 - Security (eventCategory)
 - event IDs*
 - Status (eventCategory)
 - event IDs*

CISCO CONFIDENTIAL

- Topology (eventCategory)
 - event IDs*
- Configuration (eventCategory)
 - event IDs*
- Service (eventCategory)
 - event IDs*
- Informational (eventCategory)
 - event IDs*
- Control (eventCategory)
 - event IDs*
- Event Severity
 - Critical (eventSeverity)
 - Major (eventSeverity)
 - Minor (eventSeverity)
 - Informational ((eventSeverity)

Using the EDS Atom Definition File

Atoms are defined in an atom definition file. A utility is provided to parse the atom definition file, populate the atom service, and (optionally) generate Java and C++ code that defines “constants” to be used in creating the event sources and consumer applications. The keywords and format of the atom definition file are as follows:

```
INCLUDE atom_defintion_file
atomname
LOCALE locale_name
MESSAGE "user readable message"
DESCRIPTION "some description"
= { parent_atomname atomnumber}
```

where:

- *atomname* is the assigned atom name
- *atomnumber* is the assigned atom number
- *parent_atomname* is the name of the parent atom (to define the hierarchy)
- *atom_definition_file* is the file that defines the parent atom
- *locale_name* is one of the following keywords, used to define the locale of this message:
 - INCLUDE - include this atom definition file for defining parent names
 - LOCALE - locale value for this message
 - MESSAGE - the user understandable message
 - DESCRIPTION - description of this message

The atom definition file for the base atom definitions is BaseAtomDef.atom.

The main Java class that parses the atom definition file is com.cisco.cmf.eds.atom.ParseMain. It has the following usage statement:

CISCO CONFIDENTIAL

```
java com.cisco.cmf.eds.atom.ParseMain [-J] [-C] [-U [-R]] [-I include_dirs ...] -F
atom_definition_file.atom -h
```

where:

- -J generates a Java file.
- -C generates a C++ include file.
- -U updates the atom service.
- -R replaces the atom definition in the atom service (if it exists).
- -I looks in the include_dirs for include files.
- -F specifies the file that contains the atom definition file (atom_definition_file).
- -h prints a usage message.

The generated Java file defines attributes for each atom in the atom definition file. These are defined as follows:

```
public class atom_definition_file {
    public static final atomname = #;
}
```

Java “constants” of the form `atom_definition_file.atomname` can be used in Java event sources and in event consumers.

The generated C++ include file defines attributes for each atom in the atom definition file. These are defined as follows:

```
class atom_definition_file {
    public:
        enum atom_definition_fileEnum {
            atomname = #
        };
};
```

C++ “constants” of the form `atom_definition_file::atomname` can be used in C++ event sources.

The atom service uses Java Resource Bundles to access the atom information. The program receiving the event should use the `getBundle()` static method, passing in as the base name “com.cisco.nm.cmf.eds.atom.EventAtomBundle”. Then, turn the atom integer value into a string, and use that as the key in the `getString()` and `getStringArray()` instance methods of the bundle. The following keys values are defined:

```
key - get message, if the message is not defined, return the key value
key.name - get atomname
key.description - get atom description
key.fields - get string array of keys defined under this key in the hierarchy
```

**Note**

`key` is the string value of the integer passed in the event. `key.name` is the key value with the string “.name” appended on the end (ie. `key+”.name”`). `key`, `key.name` and `key.description` return strings and `key.fields` returns a string array.

See JDK information for exact method syntax and exceptions thrown for using Resource Bundles.

CISCO CONFIDENTIAL

Using the Atom Service Executables

The main Java class for the atom service is `com.cisco.cmf.eds.atom.Main`. It has the following runtime options:

```
java com.cisco.cmf.eds.atom.Main [-D dir_name] [-h]
```

where:

- `dir_name` is the name of directory where the atom service stores its data. It is stored in a file called `atom.src`. If no directory is specified, the directory where the atom service is started is used.
- `-h` prints usage message.

The atom service stores its data in a file called `atom.str` in the atom service's working directory. If the store file is not there, or if the atom service has a problem reading the store file, all atom definition files (files that end with `*.atom`) are read in to initialize the service.

When a new network management application is installed, it should copy its `*.atom` definition file into the atom service working directory. It should then make sure the atom service is running and then run the atom parse program using the `-U` and if necessary, the `-R` option to update the atom service with the new atoms used by the application.

Using the EDS Java Interface Classes

Java classes are provided for both the event source and event consumer to interface with EDS. These classes are defined in the `com.cisco.cmf.eds.system.*` package. They are as follows:

```
public final class EventSource {
// Public Constructor
public EventSource(String name);
public EventSource(String name, String hostname);
public EventSource(String name, int queuesize);
public EventSource(String name, String hostname, int queuesize);
public EventSource(String name, EventSourceCallbackInterface callback);
public EventSource(String name, int queuesize,
EventSourceCallbackInterface callback);
public EventSource(String name, String hostname, int queuesize,
EventSourceCallbackInterface callback);
// Public Instance Variables
public boolean init;
// Public InstanceMethods
public EventUID sendEvent(EventBase event);
// send event
public EventUID sendEventNoBlock(EventBase event)
throws QueueFullException;
//send event, do not block
//if queue is full, throw
// exception if queue is full
public void setCallback(EventSourceCallbackInterface callback);
// set callback interface
public void deleteConnection();
// delete connection to
// EDS
}

public interface EventSourceCallbackInterface {
public void eventSent(EventBase event, Exception ex);
// this event has been sent if the
// event was sent, ex was null if
// there was an exception and this
```

CISCO CONFIDENTIAL

```

// event was unable to be sent, ex
// will not be null
}
public final class FilteredEventConsumer {
// Public Constructor
public FilteredEventConsumer(String name);
public FilteredEventConsumer(String name, EventCallbackInterface
Eventcallback);
public FilteredEventConsumer(String name, String hostname, EventCallbackInterface
eventcallback);
// Public Instance Methods
public void setCallback (EventCallbackInterface eventcallback);
// used to define java callback
// interface
public void setClassInstance (EventFilterInterface object) throws
com.cisco.nm.cmf.eds.exceptions.InvalidInterface, java.io.IOException;
// use to define the filter class
// instance
public void setClassSource(String class_name, String class_source) throws
com.cisco.nm.cmf.eds.exceptions.InvalidInterface,
com.cisco.nm.cmf.eds.exceptions.CompilerError;
// used to define the filter class
// source
public void setQueryFilter(String filter_string) throws
com.cisco.nm.cmf.eds.exceptions.InvalidInterface,
com.cisco.nm.cmf.eds.exceptions.CompilerError,
com.cisco.nm.cmf.eds.exceptions.QueryParseException;
// used to define a "query" to be
// used to evaluate events for
// possible interest by this
// consumer
public void setFilterName(String filtername) throws
com.cisco.nm.cmf.eds.exceptions.FilterNameNotFound;
// used to set a named filter to be
// used to evaluate events for this
// consumer
public void setEventFilter(EventFilter filter) throws java.io.IOException;
// used to set a filter straight from
//the event filter repository
public void deleteConnection();
// delete connection to EDS
}
public interface EventCallbackInterface {
public void eventReceived (EventBase event, EventData s_event);
// after the FilteredEventClass
// has received an event, it calls
// this method of the registered
//callback interface
}
public interface EventFilterInterface {
public boolean evaluateEventData(EventBase event);
// the filter class must implement
// this interface. The consumer
// connector will call this method
// for every event received
}

```

CISCO CONFIDENTIAL

Registering EDS Application Events

All event source applications need to register the events they plan on sending. This registration allows a more detailed and complete filtering of events generated in the system. The event sources need to instantiate a “dummy” copy of events it plans on sending. An exhaustive list of events is required that encompasses all the variations of Java event classes, Event Categories, Event IDs, Event Severities, Resource Types and Resource ID Types.

This is the Java class that interfaces to the event repository:

```
public class EventHash extends java.util.Hashtable{
//Public Constructor
    .public EventHash();
//Public Instance Methods
    .public Object put(EventBase event);
//add “dummy” event to event repository
}
```

To store and access events in the repository, an instance of the EventHash class must be created. The defined hash methods can then be used to access the events. If the application is going to store the events in the repository, it must use the defined put() method above.

Using the EDS Trap to Event Service

EDS includes a generic trap receiver that receives SNMP traps and then acts on those traps. The trap receiver supports receiving SNMP traps from a defined port (the default is the SNMP trap port 162), or by connecting to an NMS product . The connection to the NMS is via a native interface using the Java Native Interface protocol. The connection to the NMS is dependent upon the types of APIs provided by the NMS for receiving event information.

The supported platforms are:

- HP OpenView on HP/UX
- HP OpenView on Solaris
- HP OpenView on NT
- NetView on AIX
- NetView on NT
- SunNet Manager
- MicroSoft Trap Service

Using the EDS Trap Receiver Framework

Upon startup, the Trap Receiver Framework loads a trap receptor that is responsible for connecting to an NMS and receiving the traps. The trap receptor is used to identify the source for the incoming traps. Once loaded, the trap receptor receives traps and passes them onto the main trap receiver component. The trap receiver then determines what actions to take upon receipt of the trap.

The Trap Receiver can receive up to 2,000 traps within 30 seconds and process them all within a 200-second window. In conjunction with the TrapToEDS action, the generated events can be produced at a rate of 10 events per second in sustained mode. Note that performance numbers will be gated by the rate at which the NMS provides the traps to the Trap Receiver Framework.

CISCO CONFIDENTIAL

The Trap Receiver ties into other standard EDS utilities, including:

- The standard debug/logging facility for logging of trace messages
- The CWCS Daemon Manager, for graceful startup/shutdown.

The following topics explain the Trap Receiver Framework:

- [Using the Trap Receptor](#)
- [Using the Trap Receiver Configuration File](#)
- [Using TrapInclude/TrapExclude Statements](#)
- [Creating Trap Actions](#)
- [Matching Trap Records](#)
- [Using the TrapToEDS Converter](#)
- [How the TrapToEDS Conversion Table is Used](#)
- [Using the TrapLaunch Action](#)
- [Using the TrapEcho Action](#)
- [Setting Trap Receiver Properties](#)

Using the Trap Receptor

The trap receptor provides the interface for communicating with specialized NMS products and receiving traps. The trap receptor is extensible, so other NMS products can be integrated at a later time. Specialized TrapReceptor classes are provided to communicate with many different types of trap sources (such as HP Openview on trap port 162).

Loading of the TrapReceptor often involves loading native libraries for interfacing to proprietary NMS trap APIs.

The TrapReceptor class must extend the following class:

```
public abstract class TrapReceptor {
public abstract void startListening(); // start the receptor
public void registerListener(TrapListener listener);
}
```

The TrapListener is an interface provided between the receptor and the remainder of the Trap Receiver Framework. The TrapReceptor, upon receiving a trap from the NMS (or any source), will make appropriate calls on the TrapListener to send the trap.

The TrapReceptor to be loaded is determined via a variable in the TrapReceiver's property file. Since loading trap receptors typically involves loading native libraries, any modification of the trap receptor requires a restart of the trap receiver.

The properties file can be configured to define the behaviour that the TrapReceptor should perform if the NMS is not running or has stopped sending events. The default is that the TrapReceptor will enter a retry/delay state while attempting to connect to the NMS. Once the retries have been exhausted, the TrapReceiver will terminate.

CISCO CONFIDENTIAL**Using the Trap Receiver Configuration File**

The trap receiver reads a configuration file that identifies the actions to be performed upon receipt of a trap. The format identifies the actions to be performed, and for those actions, the TrapBlock defining the traps to send to that action. A TrapBlock specification of “-“ indicates that all traps should be sent. Specifying a TrapBlock on an action line that does not exist will result in an error, and the action line will be discarded. [Example 20-1](#) shows a sample trap receiver configuration file.

Example 20-1 Sample Trap Receiver Configuration File

```
# Action> <ActionName> <TrapBlock> <Action Class> <Args>
#
Action TrapToEDS EDSTraps com.cisco.nm.cmf.eds.trap.TrapToEDS.class
Action MyApp MyAppTraps
com.cisco.nm.cmf.eds.trap.TrapLaunch.class/usr/bin/myApp $1 $2 $3
Action TrapEcho com.cisco.nm.cmf.eds.trap.Echo.class
#
# TrapBlock <ActionName> {
# TrapInclude <trapName> <oid> <generic> <specific>
# TrapExclude <trapName> <oid> <generic> <specific>
# }
#
# Traps to send to the TrapToEDS action
# This says to send all traps, but exclude any traps that matches an EDS trap.
TrapBlock EDSTraps {
TrapInclude AllTraps 1.3.6.1.4.1.* * *
TrapExclude EDSTrap 1.3.6.1.4.1.9.1.1.1.1 6 1
}
# traps to send to the MyApp action
TrapBlock MyAppTraps {
TrapInclude MyTrap .1.3.6.1.4.1.9.2.2.2.2 6 100
}
# All traps will be sent to the TrapEcho action because it has no TrapBlock specification.
```

Using TrapInclude/TrapExclude Statements

TrapInclude and TrapExclude statements in the Trap Receiver configuration file identify the traps to be passed to an action. For a trap to be passed to an action, the trap must satisfy the equation:

(Match any TrapInclude statement) AND NOT (Match any TrapExclude statement)

The configuration file format allows for easily extending the TrapReceiver’s actions without affecting other registered actions.

The order in which the TrapInclude and TrapExclude statements are provided in a trap block has no bearing on whether a received trap will match (it will match according to the above formula). However, it can affect performance. When a trap is received, it is compared against the TrapInclude statements in sequential order. When a TrapInclude statement matches, the remainder of the TrapInclude statements are ignored. After a match for a TrapInclude has been determined, a similar match in sequential order is performed against the TrapExclude statements. If a match is found for a TrapExclude, the trap is discarded, and no further matching is performed.

CISCO CONFIDENTIAL**Creating Trap Actions**

Actions are Java classes that perform specific functions based upon receipt of a trap. All actions must extend the following class:

```
public interface TrapBaseAction {
    public void processTrap(TrapPDU trap,
        String args[]);
}
```

When a trap is passed to an action, a list of translatable “dollar” variables, (\$), can be specified as arguments to the action. These variables are provided as a convenience for processing information contained within the trap. The specification for the variables is modeled after the \$-vars used in HP Open View. Strings that do not start with a \$, or that start with a \$ but do not match the list of variables shown in [Table 20-1](#) and [Table 20-2](#), are passed verbatim to the processTrap action command.

The notation \$n implies the Nth variable contained within the trap, where 1 represents the first variable contained within the trap.

Table 20-1 *Trap Information Variables*

Variable	Description
\$#	Number of variables in the variable binding list
\$*	All variable bindings in the format of: [1] name (type): value [2] name (type): value ...
\$n	The nth variable’s value, printed as a string
-\$n	Print the nth variable as: [n] name (type): value
+\$n	Print the nth variable as: name: value

Table 20-2 *Trap Header Information Variables*

Variable	Description
\$A	Print the Agent address from the trap as a hostname, if possible. Otherwise, print the IP address.
\$a	Print the Agent address as an IP address.
\$C	Print the community string contained within the trap.
\$E	Print the Enterprise OID as a translated oid name, if possible.
\$e	Print the Enterprise OID as a dotted decimal string.
\$G	Print the Generic trap number.
\$S	Print the Specific trap number.
\$T	Print the trap’s SysUptime timestamp.

At startup, the trap receiver creates a single instance of each action to process all trap requests; it *does not* create a new action object as each trap is received. A single action object will be created at startup, and the action’s processTrap() method will be invoked to process the trap. This facilitates actions, such as TrapToEDS, that need to open a persistent communication channel for the passing of event information.

CISCO CONFIDENTIAL

A TrapAction may be loaded multiple times if it is associated with unique actions in the configuration file. For instance, the TrapLaunch.class listed above provides unique, separate TrapLaunch functions for invoking separate commands.

Matching Trap Records

When a trap is received, it is compared against the TrapInclude and TrapExclude statements. For a trap to be passed to an action, the trap must match at least one TrapInclude statement, and it must *not* match any TrapExclude statements.

Wildcards (the '*' character), can be used at the end of an OID value, or in place of generic or specific trap numbers. If no wildcard is specified in the OID, then the OID must match exactly.

Using the TrapToEDS Converter

The TrapToEDS Converter action is an EDS Event Source that will convert an SNMP trap into an EDS event. A new event, called TrapEvent, will be created to contain the trap information. The TrapEvent will then be sent to EDS for distribution to any registered event consumers. The conversion of the trap to the TrapEvent is performed as shown in [Table 20-3](#).

Table 20-3 TrapToEDS Conversion

Event Attribute	Action
EventID	See the “How the TrapToEDS Conversion Table is Used” section on page 20-14
Event Category	See the “How the TrapToEDS Conversion Table is Used” section on page 20-14
Event Severity	See the “How the TrapToEDS Conversion Table is Used” section on page 20-14
ApplicationUID	“TrapToEDSConverter”
EventCreateTime	Compute current timestamp
Resource List	See the “How the TrapToEDS Conversion Table is Used” section on page 20-14
Event Data	{trap OID, generic number, specific number, sysUpTime, Community string, Varbinds}

How the TrapToEDS Conversion Table is Used

The mappings shown in [Table 20-4](#) equate traps generated by Cisco devices to appropriate EDS events. Each arriving trap will be matched against its OID, generic number, and specific number for a match into the lookup table. When found, the corresponding EventID, Event Category, and Event Severity will be retrieved from the table. Also contained within the table lookup will be a variable number of Resource Item records defined for each trap. These records will be used for the assignment of the ResourceList contained within the event. Each ResourceItem will consist of a 3-tuple that will contain:

- ResourceType
- Resource IDType
- Resource IDValue

CISCO CONFIDENTIAL

The Resource Type and Resource ID Type are integer values that map to atoms created with the atom service. The Resource ID Value is a string value that can be specified via a literal string, or the \$-variables, to retrieve information from the trap.

By default, every event has one resource item with a Resource Type equal to Device. If one is not specified in the table lookup, then one will be provided with the following assignments

- Resource Type = Device
- Resource ID Type = IP Address
- Resource ID Value = IP address contained within the trap

To further support complete control over the conversion of a trap into a specific event, an alternative format can be specified in the lookup table, where the name of a conversion class is specified. This class can be loaded and used for converting the trap to an event upon matching an OID, generic number and specific number. A single instance of this class will be created, and will be called for converting all traps to events that match its trap signature. This class must extend the following class:

```
public interface class TrapConverter {
public EventBase convertToEvent(TrapPDU trap);
}
```

If the trap is not found within the lookup table, then the following algorithm is used:

```
EventID:
Coldstart Trap:BaseAtomDef.Coldstart_Trap
Warmstart Trap:BaseAtomDef.Warmstart_Trap
Linkdown Trap:BaseAtomDef.LinkDown_Trap
Linkup Trap:BaseAtomDef.LinkUp_Trap
Auth Trap:BaseAtomDef.Authentication_Trap
EGP Neighbor:BaseAtomDef.EGPNeighborLoss_Trap
Enterprise Trap:BaseAtomDef.Enterprise_Trap
Event Category:BaseAtomDef.SNMPTrap
Event Severity:BaseAtomDef.EventSeverity_Informational
```

Table 20-4 TrapToEDS Mappings

Trap Name	OID Value	Generic number	Specific Number	Event ID	Event Category	Event Severity	Resource Item #1	Resource Item #n
Reload Trap	Cisco	6	0	ReloadTrap	SNMP Trap	Informational	3-tuple	3-tuple
TCP Conn Close	Cisco	6	1	TcpConnClose	SNMP Trap	Informational	3-tuple	3-tuple

Using the TrapLaunch Action

The TrapLaunch action allows launching a separate application based upon receipt of a trap. The first argument to the action will be the name of the application to be invoked. All of the arguments are passed on the command line to the application. The TrapLaunch action will be invoked a single time, (as are all trap actions). However, the nature of the TrapLaunch is to invoke a command upon receipt of a trap. Therefore, for each trap sent to the TrapLaunch action, the action will invoke a new command.

Using the TrapEcho Action

The TrapEcho action takes the incoming trap and sends it to a specified trap port. This allows for the operation of the TrapReceiver with other products that want to receive traps on a given port.

CISCO CONFIDENTIAL**Setting Trap Receiver Properties**

You can modify the Trap Receiver properties shown in [Table 20-5](#).

Table 20-5 **Settable Trap Receiver Properties**

To set the	Modify this property
Trap port for the UDPTrapReceptor to listen on	trap_port=162
Name of configuration file	trap_config_file=filename
Trap port for the TrapEcho command to send traps	trap_echo_port=5000
Class that identifies the TrapReceptor to be loaded	trap_receptor=com.cisco.nm.cmf.eds.trap.SomeTrapReceptor
Number of retries to perform for connecting to NMS, or UDP port	trap_receptor_connection_retry=10000
Number of seconds to delay between retries	trap_receptor_connection_delay=300

Using the Generic Consumer Framework

The Generic Consumer Framework (GCF) provides a mechanism for providing generic event-consumers with a pluggable interface for receiving events. Using the Named Filter API, you can register a set of event consumers with the GCF. Generic event consumers use a different named filter for the reception of their events.

While your application can produce a standard event consumer for receiving EDS events, doing so requires you to write a filter mechanism and a specification for the associated filter. Using the GCF allows you to write a generic event consumer that only needs to focus on event processing; the filter work is part of the Framework. This interface also allows you to easily update the filter to be associated with a consumer and the events that it should receive.

The following topics explain the GCF:

- [Using the GCF Configuration File](#)
- [Using the GCF Admin Display](#)
- [Creating Generic Consumers](#)
- [Using the Event to Trap Converter with Generic Consumers](#)

Using the GCF Configuration File

The GCF configuration file specifies the generic consumers and the named filters registered for each consumer. The GCF will register with EDS on behalf of the consumers and then pass the appropriate events to the consumers.

A GUI interface exists to aid in updating the TrapReceiver's configuration file. Also, an IDL interface to the TrapReceiver is provided for receiving, and updating named filter information.

```
#
# Sample Generic Consumer Framework Configuration file
#
# Action <ActionName> <NamedFilter> <Generic Event Consumer> <Args>
Action EventToTrap - com.cisco.nm.cmf.eds.trap.EventToTrap.class
```

The use of the string “-” as a named filter equates to a filter that indicates to receive all events.

CISCO CONFIDENTIAL

Using the GCF Admin Display

The GCF administrative display permits configuring the named filters to be associated with generic consumers. It also displays the description of the selected filter. These features aid the administrator in choosing the correct named filter.

The GCF Admin Display exists as an HTML interface. Upon selecting appropriate information, the GCF Admin display interfaces to web servlets communicating on the backend for updating the GCF information.

Creating Generic Consumers

Generic consumers are consumers to be registered with the GCF. Generic consumers have no knowledge of the filters associated with them. After startup, the GCF passes events to those generic consumers whenever the event passes their associated filter.

All generic consumers must extend the following class:

```
public interface GenericEventConsumer{
    // Pass the event to the consumer.
    public void processEvent(EventBase event, String args[]);
}
```

When passed an event via the processEvent() method, a list of \$-translatable variables can be specified as arguments to the generic event consumer. These variables permit information to be obtained from the event without having to access the event object itself.

Using the Event to Trap Converter with Generic Consumers

The Event to Trap Converter (ETC) service plugs into the GCF, allowing generic consumers to receive EDS events. The named filter associated with the service determines the events that the EventToTrap service receives. Once the events have been converted into a trap, the traps will then be forwarded to a network management station via a destination hostname and port.

Upon startup, the ETC will read a startup file containing hostname and destination port settings for forwarding traps. If no port is specified, the standard SNMP trap port, 162, is assumed.

When an event passes the ETC filter, the event's toTrap() method is called to return a TrapClass object for the event. The TrapClass object supports all features needed to convert the event into a trap:

```
public abstract class TrapClass {
    public abstract String getEnterpriseOid();
    public abstract int getGenericTrapNumber();
    public abstract int getSpecificTrapNumber();
    public abstract String getCommunity();
    public abstract String getIPAddr();
    public abstract long getSysUpTime();
    public abstract SnmpVarbindList getVarbindList();
}
```

Upon receiving the TrapClass object, the ETC will call the appropriate methods to create the TRAP PDU and then send the PDU to the specified NMS hosts. By overriding the TrapClass object that is returned via the toTrap() method for an event, an event writer can define its own Event-To-Trap conversions.

In the case where an event does not provide a conversion, a DefaultTrapClass() converter object will be used. Your application can extend the DefaultTrapClass() if you want to modify some aspect of its functionality. The base DefaultTrapClass conversion produces the results shown in [Table 20-6](#).

CISCO CONFIDENTIAL**Table 20-6 DefaultTrapClass Conversion Results**

Attribute	Value
OID	1.3.6.1.4.1.9.9.127.2.0
Generic trap number	6
Specific trap number	Critical event: 1 Major event: 2 Minor event: 3 Informational: 4
Agent Addr	Inspect resource list for an IP address, or Hostname, otherwise IP address of hostname where Converter resides.
Community string	Default from EDS property file
SysUpTime	Uptime for Event Consumer
Varbind 1	Event ID number
Varbind 2	Event ID name
Varbind 3	Event Category number
Varbind 4	Event Category name
Varbind 5	Event Created Time
Varbind 6	Event Sent Time
Varbind 7	Application name
Varbind 8	Event Class name
<Repeated>	Event Resource Information
<Repeated>	Unique Data name/value

The EventToTrap converter reads a property file containing various startup parameters. The property file consists of the following:

```
# list of hosts to receive trap information
gcf_trap_receivers=hostname:port,hostname:port

# community string to use within traps
community=public
```

Using EDS to Publish Events

The CWCS Job and Resource Manager (JRM) uses EDS to publish events of interest to CWCS-based applications. These events belong to the “status” event category (EventCategory_Status). The EDS event and resource atoms are listed in com.cisco.nm.cmf.jrm.JrmEdsAtomDev.java.


The following topics explain how EDS publishes events and how to receive them:

- [About the EDS-Published Event Types](#)
- [About the EDS-Published Severity Codes](#)
- [Registering Your Application with EDS](#)

CISCO CONFIDENTIAL**About the EDS-Published Event Types**

Table 20-7 summarizes the typical JRM-related event types published via EDS.

Table 20-7 *JRM Events Published Via EDS*

Event Type	Description
Job-related	<p>All job events are the instances of the <code>com.cisco.nm.cmf.jrm.JobEvent</code> class. The event's resource list always contains two resources: <code>ResourceList_Job_Type</code> and <code>ResourceList_Job_Id</code>. The event's unique data contains four members:</p> <ul style="list-style-type: none"> • Job <code>szProgress</code> field at the time the event was recorded. <p> Note For the approve/reject events <code>szProgress</code> will contain an approver's comments.</p> <ul style="list-style-type: none"> • Job <code>run_state</code> field. • Return code (for <code>EventJobEnd</code> event). • Approver name (for <code>EventJobApprove</code> and <code>EventJobReject</code>).
Lock-related	<p>Lock/unlock events do not have unique data. The resource list contains two resources:</p> <ul style="list-style-type: none"> • <code>ResourceList_Address_Hostname</code> contains the resource name. • <code>ResourceList_Job_Id</code> contains the job name.
Process- end	<p>Process end events are instances of the class <code>com.cisco.nm.cmf.jrm.DaemonEndEvent</code>. The resource list contains a single resource of type <code>ResourceList_Job_Id</code>. The event's unique data contains the return code and signal code of the process.</p>

About the EDS-Published Severity Codes

Table 20-8 summarizes the JRM event severity codes used with EDS-published events.

Table 20-8 *JRM Event Severity Codes Published Via EDS*

Severity Level	Codes
EventSeverity_Information	EventLock EventUnlock EventJobStart EventJobEnd (if completion code is <code>RUNST_Succeeded</code>) EventJobCancel EventJobApprove EventJobReject EventDaemonEnd
EventSeverity_Minor	EventJobEnd (if completion code is <code>RUNST_SucceededWithInfo</code>)
EventSeverity_Major	EventJobEnd (if completion code is <code>RUNST_Failed</code>) EventJobLaunchFailed

CISCO CONFIDENTIAL

Registering Your Application with EDS

To send and receive EDS events, you must subscribe to the events you want to view.

The `SampleEventConsumer.java` file, located in the `CodeSamples` directory on the CWCS SDK CD, shows how to subscribe to EDS events. This example shows:

- How to create an instance of a filter
- How to create the source code for a filter

It also displays two lists:

- The top list displays events received from one filter.
- The bottom list displays events received from a second filter.



CISCO CONFIDENTIAL

CHAPTER 21

Using the Installation Framework

CWCS supplies several tools for application installation, uninstallation, and patching. Because each platform has its own set of standards, formats, and issues, you will need different tools. But the basic installation concepts are similar.



Note Cisco encourages developers to leverage the installation framework discussed in this chapter. The tools for user interface and installation integration are free and will save your team time.

The installation framework allows you to consider dependencies and features such as uninstallation, which makes it easier for your package to work like other network management packages. The installation framework can help ensure that prerequisites such as version dependencies are set and followed.

Cisco recommends that you use both the build environment and the installation framework. The build environment is tuned to enforce Cisco procedures and policies. The installation framework facilitates a common look-and-feel and the required CWCS bundle behavior. However, the installation framework does *not* require using the build environment.

The following topics describe the installation framework and associated processes:

- [About the Installation Framework](#)
- [Getting Started with the Installation Framework](#)
- [Using the Installation Framework](#)
- [Windows Installation Reference](#)
- [Solaris Installation Reference](#)

For information on installing CiscoWorks Common Services itself, refer to the “[Installing CWCS](#)” section on page 5-2.

About the Installation Framework

This section discusses the following topics:

- [What’s New in This Release](#)
- [Understanding the CWCS Installation Framework](#)
- [Understanding Installation Team Responsibilities](#)
- [Understanding Developer Responsibilities](#)

CISCO CONFIDENTIAL

What's New in This Release

New or changed features in this release of the CWCS installation framework include:

- Express mode is no longer supported. The only installation modes are Typical and Custom.
- Remote upgrade is no longer supported. This has been replaced by the CWCS migration framework.
- The installation framework is now compatible with the new CWCS licensing framework, vastly simplifying the work applications need to do (see the [“Providing Licensing Information During Installation”](#) section on page 21-22).
- Advice on reducing installation time has been added (see the [“Reducing Windows Installation Time”](#) section on page 21-36).
- Solaris installation now supports limited workflow customization (see the [“Customizing the Installation Workflow on Solaris”](#) section on page 21-104).
- The new command `pkgchk` permits package verification before or after Solaris installs (see the [“Verifying Packages on Solaris”](#) section on page 21-105).

Understanding the CWCS Installation Framework

The CiscoWorks Common Services (CWCS) runtime environment provides shared functionality for applications when they are running. It provides services such as scheduling, process management, and database storage and retrieval so that your applications can perform their tasks. The CWCS installation framework provides shared functionality for individual applications so that they can install on different kinds of systems with similar interfaces and consistent install semantics.

The goals of the installation framework are:

- To isolate the installation process and tools (as much as possible) from the application so the application only needs to worry about application-specific install issues.
- To allow applications to specify hardware and software requirements as well as dependencies, regardless of the target operating system.
- To support the orderly uninstallation of applications.
- To facilitate a common user experience for all CiscoWorks products.
- To allow the user to install the package easily, regardless of the target operating system or application being installed.
- To update package file ownership settings during the build/installation on Solaris.

The installation framework provides an installer, an uninstaller, and a set of functions that facilitate installation-specific tasks.

The installation team and the developer have separate roles to play in the process of adding packages for installation:

- The installation team is responsible for ensuring that the main installation script and build tools (the installation framework) are available and working (see the [“Understanding Installation Team Responsibilities”](#) section on page 21-3).
- The developer is responsible for specifying what objects should be installed, where they should be installed, and how they should be installed (see the [“Understanding Developer Responsibilities”](#) section on page 21-3).

CISCO CONFIDENTIAL

Understanding Installation Team Responsibilities

The installation team is responsible for the main installation script. This script is the same for CiscoWorks, Resource Manager Essentials, Campus Manager, and other CWCS-based products. It does not change (on Windows platforms, the main installation script can be customized; for details, see the “[Customizing the Installation Workflow for Windows](#)” section on page 21-76). It is important to use the installation framework to allow for the following:

- **Component sharing:** Enables several packages within the product to share functionality without installing packages more than once.
- **Dependencies:** Includes source and target, version, backward compatibility, and uninstallation.
- **Optional components:** installation of optional components based on requirements or dependencies.
- **Development:** Allows individual business units or third-party developers to produce components.
- **Delivery:** Allows individual business units or third-party developers to release their products as part of a major or minor release or drop-in.
- **Maintenance:** Provides for backward compatibility, upgrades, and patches that are consistent.
- **Security:** Allows CiscoWorks applications to use the same file ownerships and permissions schema as CWCS (see the “[Understanding and Implementing the casuser](#)” section on page 21-21).

The installation team is also responsible for the tools that create the installable CD image, which are discussed in the “[Getting Started with the Installation Framework](#)” section on page 21-4. Developers outside of NMTG can use their own build tools and build environment to produce their executable files, but should use the installation tools to build CDs. NMTG developers should use internal build and installation tools.

For assistance with running the installation framework software, reporting problems, or questions about the software:

- Refer to the Installation Framework team web site:
https://mco.cisco.com/ubiapps/portal/servlet/EEngPortalDispatcher?EVENT=ProjectPortalDisplayEvent&portal_id=1975
- Contact: cmf-install-dev@cisco.com

Related Topics

See the:

- “[Understanding Developer Responsibilities](#)” section on page 21-3.
- “[Understanding the CWCS Installation Framework](#)” section on page 21-2.

Understanding Developer Responsibilities

The CWCS installation framework is provided by the installation team, but can be used in different ways depending on your requirements:

- If you are a developer working outside NMTG (either in another business unit or as a third-party partner), you should use the installation framework to enable better integration, improve look and feel, and take advantage of features such as uninstallation.
- If you are a developer working inside NMTG, in addition to using the installation framework you must use the automated build processes to create protopackages. Your build process has the build tools to automatically create protopackages.

CISCO CONFIDENTIAL**Related Topics**

See the “[Understanding Installation Team Responsibilities](#)” section on page 21-3.

Getting Started with the Installation Framework

The Installation Framework is supported on Windows and Solaris platforms. It follows the server platform-support requirements for CWCS, but can be customized for other Windows and Solaris platforms. For other platform enquiries, contact the installation team (cmf-install-dev@cisco.com).

Main installation script and build tools for the CWCS installation framework are available from the CWCS 3.0 SDK Portal at https://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=2537.

This section discusses the following topics:

- [Third-Party Tools for Installation Framework](#)
- [Understanding Install Component and Image Structures](#)
- [Preparing Installation Protopackages](#)

Third-Party Tools for Installation Framework

The following tables describe the required third-party tools for Windows and Solaris:

Table 21-1 *Third-Party Tools for Windows*

Name	Description
InstallShield 5.53 Professional only	Installs packages and files. Purchase from InstallShield Software Corporation. The Maintenance Pack 3 for the InstallShield 5.5 is required.
InstallShield PackageForTheWeb version 4.x	Packages CD image into the self-extracting .EXE file. Download from InstallShield Software Corporation. See http://www.installshield.com .
MKS Toolkit 6.1 or higher	Builds an image from protopackages. Purchase from the Mortice Kern Systems Inc., Tel: (519) 884-2251, http://www.mks.com

Table 21-2 *Third-Party Tools for Solaris*

Name	Description
Perl version 5.5 or higher	Builds an image from protopackages. Shareware/public domain software located at http://www.perl.com/pub .

Understanding Install Component and Image Structures

CWCS software is divided into a set of components that reflect the logical structure of the software. In this document, the following terms are used to describe the package structure:

- package—The smallest piece of software processed by the installation framework. Packages are defined by a set of properties. All runtime files belong to packages.

CISCO CONFIDENTIAL

- **installable unit**—A group of packages that are always installed or uninstalled simultaneously. A package can only belong to one installable unit (or can belong to no installable unit). An installable unit is also defined by a set of properties; on Solaris it cannot directly include runtime files.
- **suite**—A group of packages or installable units that define a product or an application, usually as required by marketing.
- **component**—A package, installable unit or suite.
- **protopackage**—A protopackage is a tar file that contains a component of the product. This component has a name, version, and other properties. It also contains the runtime and installation-related information. A protopackage is used to prepare an input to the installation framework. Typically, protopackages are created automatically by the build tools after compilation, linking, etc.
- **tag**—Protopackages are identified by a tag. This tag is the short internal name, which is the same as the first word in the name of the protopackage file. For example, the `cmf.runtime.tar` protopackage has this tag: `cmf`.
- **image**—A set of files that contains one or more products ready for installation. An image can be directly burnt on a CD. It contains the installer as well as all product components. Each image contains the Table of Contents file (`disk.toc`) that specifies the metadata about the image, as well as some instructions for the installer.

The following topics describe the process to follow in building installable images, and the package components and image structures:

- [Building an Installable Image](#)
- [Selecting Package Names](#)
- [Specifying Package Properties](#)
- [Understanding the Package Properties File](#)
- [Understanding Suite Properties](#)
- [Creating the Table of Contents](#)

Building an Installable Image

Use the following process to build an installable image:

-
- Step 1** Select the package name (see the [“Selecting Package Names”](#) section on page 21-6) .
 - Step 2** Specify properties for packaging (see the [“Specifying Package Properties”](#) section on page 21-6).
 - Step 3** Write scripts to specify installation requirements and enforce constraints:
 - For Windows, write InstallShield scripts (see the [“Writing Windows Scripts”](#) section on page 21-33).
 - For Solaris, write Bourne shell scripts (see the [“Writing Solaris Scripts”](#) section on page 21-90).
 - Step 4** Prepare the Table of Contents (see the [“Creating the Table of Contents”](#) section on page 21-11).
 - Step 5** Create protopackages that include the runtime files, package properties, scripts, and Table of Contents created in the previous steps (see the [“Preparing Installation Protopackages”](#) section on page 21-19).
 - Step 6** Build and debug CDs (see the [“Using Solaris Build Tools”](#) section on page 21-103).
-

CISCO CONFIDENTIAL**Selecting Package Names**

A package name, or *tag*, is a short internal name that is used programmatically. Normally it is not presented to the end user.

Windows Package Names

For Windows platforms, the package name must match the name of .pkgpr file, for example ut.pkgpr must have PKG=ut. The following table describes the Windows package name requirements.

Table 21-3 Windows Package Name Requirements

Requirements	Description
Naming Conventions	Eight character limitation, CSCO prefix is not required. For example, ut.
Package name must match pkgpr files	Required. Example, ut.pkgpr.
Names must be unique	Required, but only within CiscoWorks products. It will not collide with non-CiscoWorks products.

Solaris Package Names

For Solaris, the package name must begin with CSCO followed by up to five characters. The following table describes the Solaris package name requirements.

Table 21-4 Solaris Package Name Requirements

Requirements	Description
Naming Conventions	CSCO prefix is required, with no more than nine characters total. For example, CSCOjrm.
Package name must match pkgpr files	Not required.
Names must be unique	Required. Must be unique across all products on this platform.

Specifying Package Properties

Use the package properties file (*tag.pkgpr*) to specify the component properties for each package. The package properties file contains name-value pairs for the platforms your package supports and specifies the properties for each platform.

Any platform-specific name value pairs should be specified in the package properties file using the following line immediately before the pairs appear in the file:

```
PLATFORM_NAME:
```

**Note**

Remember to always use a colon to separate any platform names or to complete the end of a line.

The platform names are represented as:

- NT: for Windows
- SOL: for Solaris

CISCO CONFIDENTIAL

For example, any Solaris-specific name value pairs in the package property file must have the line:

```
SOL:
```

Several of them can be combined as follows:

```
SOL:NT:
```

Understanding the Package Properties File

The package properties file (<tag>.pkgpr) include several groups of properties. The following topics describe these property groups:

- [Developer-Specified Properties](#)
- [Appended and Generated Properties](#)
- [Solaris-Specific Properties](#)

Developer-Specified Properties

Table 21-5 show the package properties that developers must specify for both Windows and Solaris platforms. The image build tools will transfer these name-value pairs into the <tag>.info files. Finally, the installation process copies them into the target system.

Table 21-5 <tag>.pkgpr Files Name-Value Pairs

Name	Format	Description	Required
PKG	Solaris: CSCOxxxxx Windows: up to eight characters	Package tag. A short internal name, used programmatically. Normally invisible to the end user. For Solaris, it must begin with CSCO followed by a maximum of five characters. For Windows, it must match the name of the .pkgpr file. For example, ut.pkgpr must have PKG=ut.	Yes
NAME	String, up to 40 characters	One-line name to be presented to the end user whenever components are listed.	Yes
DESC	String	Short description (up to 1000 characters).	Yes
VERSION	X.Y, where X and Y are numbers	Version, major and minor	Yes
PATCHVER	Number	Patch level	0 by default
DEPENDS	String	Comma-separated list of components this one depends on, each in the form of <i>pkgTag major.minor.patchlevel</i> , where <i>pkgTag</i> is the package tag; <i>major</i> , <i>minor</i> , and <i>patchlevel</i> specify the version of dependency target. If a specific version of dependency target is not required, then you can omit <i>major.minor.patchver</i> . Correct: DEPENDS=cmf, xrts 1.2.1, ani Not correct: DEPENDS=xrts 1.2 (It should be DEPENDS=xrts 1.2.0.)	No
ROOTDIR	String	Alias for root directory. Default value <i>NMSROOT</i> . ¹	No

CISCO CONFIDENTIAL**Table 21-5** <tag>.pkgpr Files Name-Value Pairs (continued)

Name	Format	Description	Required
PACKAGES	String	Space-separated list of packages that belong to this installable unit. This property must be specified for installable units only.	Yes, f or installable units only.
BACKVER	X.Y.Z (where X, Y, Z are numbers)	The <i>major.minor.patchlevel</i> , where <i>major</i> , <i>minor</i> , <i>patchlevel</i> specify the version. If you do not use all three version numbers, use no version numbering. This version is backward compatible with all versions starting from the one specified by this parameter. Example: <code>VERSION=3.2, PATCHLEVEL=3, BACKVER=2.4.0.</code> If any other package depends on version 2.6 of this package, it can be installed.	No
OPTIONAL	Y	Specifies that a package is optional. An optional package can be dropped if dependencies for it cannot be resolved. It is assumed that installable unit is operational with or without optional packages.	No
SIZE	Number	Space, in megabytes, required for this package. The installer calculates a package's footprint automatically. This property provides for override. Windows calculates this automatically.	Yes, for Solaris only.
REC_RAM REC_SWAP REC_DISK REC_CPU		These properties can be set for packages or installable units but are normally set for Suites only.	No
CMFSERVICES	String	Comma-separated list used to register for specific CiscoWorks service bundle use. Possible values: System, Network, Core.	No
UNINSTALL_REBOOT	Windows only: Y or y	Indicates that the system needs to be rebooted at the end of uninstallation (if this package is uninstalled).	No

1. The installer maintains the list of root directories for all components. The value of ROOTDIR property is an alias, which can be used in installation hooks. See the "Locating the Root Directory Path Name" section on page 21-54 for more information.

Appended and Generated Properties

Other properties are also present in the package properties file. These are appended to the file during the build process or generated by the installer:



Note The properties in [Table 21-6](#) and [Table 21-7](#) appear in the <tag>.info files, but *not* in the <tag>.pkgpr files.

- [Table 21-6](#) describes the properties appended to the properties package file by the build process.
- [Table 21-7](#) describes the properties generated by the installer. Platform-specific requirements are noted where applicable.

CISCO CONFIDENTIAL**Table 21-6** Properties Appended During the Build Process

Name	Format	Description
BUILD_ID	String	Build ID in the form platform_project_date_type. For example: NT_CMF_RIGEL_19990408_0126_daily. This ID id is generated by make protopackage ¹ by combining the values of PROP_ID and PROP_WHEN from the .bprops file.
BUILD_TSTAMP	Number	Timestamp, generated from PROP_TIMESTAMP in the .bprops file.
ALIAS	String	Contains the names following the PKG/SUITE property for each platform. Contents written to the .info file. For example, dmgt.info/CSCCmd.info both contain the new property ALIAS=dmgt CSCCmd. Using a space field separator, you can determine if a .info file matches the package by examining the ALIAS property.

1. **make protopackage** is specific to the NMTG build environment.

Table 21-7 Properties Generated by the Installer

Name	Format	Description
Date	String	Date and time of installation.
PREV_VERSION	See VERSION	Version that has been installed before this one.
PREV_PATCHVER	See PATCHVER	Patch level that has been installed before this one.
I_MODE (UNIX only)	String	NEW, REINSTALL, DOWNGRADE, PATCH, or UPGRADE.
OS (UNIX only)	String	SOL, HPUX, or AIX.

Solaris-Specific Properties

[Table 21-8](#) describes Solaris properties that are not generated automatically, *but that are required*. Add these lines to your pkgpr file.

Table 21-8 Solaris-Specific Properties

Property Name	Definition	Example
CATEGORY	Type of package	CATEGORY=application
ARCH	Architecture package supports	ARCH=sparc
VENDOR	Who created the package	VENDOR=Cisco Systems, Inc.
CLASSES	For character abbreviation	CLASSES=TBD
BASEDIR	Top-level install directory.	BASEDIR=/opt

The information about prototype.header properties listed in [Table 21-9](#) is provided for developers who are not familiar with Solaris packaging tools. The prototype.header file required for Solaris packaging is generated automatically and is not to be maintained by developers. If you create your own prototype.header file, we will overwrite it.

**Note**

The properties in [Table 21-9](#) should *not* be specified manually by developers.

CISCO CONFIDENTIAL

Note Developers can define their own properties in the *tag.pkgpr* file for their own use.

Table 21-9 Properties for *prototype.header*

Name	Format	Description	Required
IU_NAME	String, up to 256 characters	One-line name presented to the end user. Used whenever components are listed.	Yes
IU_VERSION	X.Y, where X and Y are numbers	Version, major and minor.	Yes
IU_DESC	String	Short description—up to 1000 characters.	No
IU_PATCHVER	Number	Patch level.	No
DEPENDS	String	Comma-separated list of components this one depends on, each in the form of <i>pkgTag major.minor.patchlevel</i> , where <i>pkgTag</i> is the package tag; <i>major</i> , <i>minor</i> , <i>patchlevel</i> specify the version of the dependency target.	No

Name	Format	Description
I_MODE	String	NEW, REINSTALL, DOWNGRADE, PATCH, or UPGRADE
NMSROOT	String	Path name of target directory
SETUPDIR	String	Path name from which the CD installation is running

Understanding Suite Properties

Suite properties describe different bundles of installable units and/or packages. [Table 21-10](#) describes the suite properties for both Windows and Solaris.

Table 21-10 Suite Properties

Name	Format	Description
SUITE	Up to eight characters	Suite tag. Tag is short internal name, which is used programmatically. Normally it is not presented to the end user.
NAME	String, up to 40 characters	One-line name to be presented to the end user.
DESC	String	Short description, up to 1000 characters.
VERSTR	String	Version string to be presented to end user. For example, “Service Pack 3.”
DEPENDS	String	Comma-separated list of components this one depends on, each in the form of <i>pkgTag major.minor.patchlevel</i> , where <i>pkgTag</i> is the package tag; <i>major</i> , <i>minor</i> , and <i>patchlevel</i> specify the version of dependency target. If a specific version of dependency target is not required, then <i>major.minor.patchver</i> can be omitted. Correct: <code>DEPENDS=cmf, xrts 1.2.1, ani</code> Incorrect: <code>DEPENDS=xrts 1.2</code> (It should be <code>DEPENDS=xrts 1.2.0</code> .)
TAGS	String	Space separated list of packages or installable units, which belong to this suite.

CISCO CONFIDENTIAL**Table 21-10 Suite Properties (continued)**

Name	Format	Description
REC_RAM REC_SWAP	Number	Specifies the recommended size of RAM (in MB) and swap/paging file size (in MB) for this suite. The install process collects these requirements from all components, finds the maximum and verifies if the server has enough RAM and swap. This process displays a warning if these requirements are not satisfied. It is warning only. Installation will not abort. ¹
REC_CPU	Number	(Windows only.) Specifies the recommended CPU speed in MHz. The install process collects these requirements from all components, finds the maximum and verifies if the server has enough RAM and swap. The install process displays a warning if these requirements are not satisfied. It is only a warning. Installation will not abort. ¹
SSL_COMPLIANT	String. Possible values: Yes or No	Specifies whether the suite is SSL compliant. The installation process detects whether the suites are SSL compliant. The installation program does not allow the installation of the suite if the underlying Ciscoworks installation is SSL enabled. This field is also used by other components, such as Daemon (Process) Manager, and Enable/Disable SSL. The installation framework, daemon manager and other modules of CWCS check the SSL_COMPLIANT field in *.info files under <i>NMSROOT</i> /setup, only if the info files have a SUITE field. In other words, CWCS detects SSL compliance only in SUITE level information files. If an application cannot add the SUITE tag in info files, you can use the others directory under <i>NMSROOT</i> /setup. CWCS checks for SSL-compliance in all information files in <i>NMSROOT</i> /setup/others directory, irrespective of whether it contains the SUITE tag or not. The application should store such info files under <i>NMSROOT</i> /setup/others directory. Note Other CWCS modules uses the info files under <i>NMSROOT</i> /setup. So the applications can store the info files in both directories, to make use of the appropriate CWCS function. The applications and versions page will use the SSL_COMPLIANT value from info files from both directories - <i>NMSROOT</i> /setup <i>NMSROOT</i> /setup/others.

1. On PCs, the CPU speed and RAM size are often reported incorrectly by MS Windows. For example, HP Vectra reports 1 MB RAM less than it actually has. For this reason, installation will consider RAM and CPU speed sufficient if the server has 5 MB less than required.

Creating the Table of Contents

The table of contents is specified by disk.toc, an ASCII file that contains:

- The contents of the CD
- Components available on this CD
- Defaults for installation
- Release-specific information (release name, etc.)

The following is an example of a table of contents file:

```
[RELEASE]
NAME=Test Application with CiscoWorks Common Services 3.0
PRODUCT_NAME=CiscoWorks
```

CISCO CONFIDENTIAL

```
SHORT_PRODUCT_NAME=CiscoWorks
VERSTR=1.0
```

```
[COMPONENTS]
TAGS=cdone, CSCOMyApp
UNINSTALLABLE=cdone, CSCOMyApp
VISIBLE=cdone, CSCOMyApp
DEFAULT=cdone, CSCOMyApp
```

```
[ADVANCED_CHOICE_1]
ADVANCED_CHOICE_1_CONDITION=TRUE
ADVANCED_CHOICE_1_TYPE=NONE
ADVANCED_CHOICE_1_DEFAULT=1
ADVANCED_CHOICE_1_1_TEXT=Everything
ADVANCED_CHOICE_1_1_TAGS=cdone, CSCOMyApp
```

The table of contents consists of the following sections. Each section defines one or more parameters in the form of name=value pairs.

- **RELEASE**—Contains the name, description, and version string for the release (see the [“Creating the RELEASE Section” section on page 21-12](#)).
- **COMPONENTS**—Specifies additional properties for components (suites, installable units, and packages) available on the CD (see the [“Creating the COMPONENTS Section” section on page 21-13](#)). This section is mandatory.
- **Optional override sections**—Each section has a name, which is a valid component tag. Parameters in this section override those specified as component properties or provide additional properties (see the [“Creating Overrides Sections” section on page 21-14](#)).
- **ADVANCED_CHOICE**—Each section defines a component choice scenario. There can be more than one scenario; each section describes one scenario (see the [“Creating Advanced Component Sections” section on page 21-14](#)).
- **PROMPT_INSTALL_TYPE**—**Solaris only:** Specifies the installation mode (Typical or Custom) (see the [“Creating the PROMPT_INSTALL_TYPE Section” section on page 21-16](#)).
- **INSTALL_TYPE_SELECTION**—**Windows only:** Specifies the installation mode (Typical or Custom) (see the [“Creating the INSTALL_TYPE_SELECTION Section” section on page 21-16](#)).
- **BUILD**—Automatically generated at packaging time (see the [“Understanding the BUILD Section” section on page 21-17](#)).
- **REQUIRED**—Specifies the products that are required for this installation (see the [“Creating the REQUIRED section” section on page 21-18](#)).

Related Topics

See the:

- [“Preparing Installation Protopackages” section on page 21-19](#).
- [“Step 3: Prepare the Make Image on Solaris” section on page 21-103](#).
- [“Customizing the Installation Workflow for Windows” section on page 21-76](#).
- [“Solaris Getting Started Example” section on page 21-106](#).

Creating the RELEASE Section

The RELEASE section of the table of contents file, disk.toc, contains the properties of this release. This data is displayed to the end user during installation and is included in history and installation log files. It is not included in the component database.

CISCO CONFIDENTIAL**Table 21-11 Table of Contents File—Release Section**

Name	Format	Description
NAME	String, less than 20 characters	Name of this release. Included in the welcome dialog.
DESC	String, less than 100 characters	Description. Can provide additional information about the CD.
VERSTR	String	Version string. This can be something like “2.5 Eval” or “Maintenance release.” This is displayed next to the NAME and can provide additional information about CD.
README	String.	Optional for Windows only. Path name for this file is displayed at the end of the installation if the end user elects to display the readme file. For example, README= <i>NMSROOT</i> :\htdocs\help\netc\ref_config.html

Creating the COMPONENTS Section

The COMPONENTS section in the table of contents file (disk.toc) lists the major components available on this CD and the user information about these components. This section is mandatory.

Table 21-12 Table of Contents File – COMPONENTS Section

Name	Format	Description
TAGS	Comma-separated list	Comma-separated list of components. Each value is a tag of component. Components are defined by corresponding property set (see the “Specifying Package Properties” section on page 21-6” and the “Creating the Table of Contents” section on page 21-11). The installer starts processing the CD from this list. For each component listed, it looks for the subcomponents defined by the PACKAGES property. It is not necessary to include all packages/installable units in this list. It requires only the roots of component hierarchy. Mutually exclusive with PATCH_TAGS.
PATCH_TAGS	Comma-separated list	Comma-separated list of components. Each value is a tag of component. Components are defined by a corresponding property set (see the “Specifying Package Properties” section on page 21-6” and the “Creating the Table of Contents” section on page 21-11). This list defines the top level components available on CD and also turns on the patch mode of installation. This information is used instead of the TAGS for patching as described in “Handling Patches” section on page 21-27.
VISIBLE	Comma-separated list	Comma-separated list of components, that are exposed to the end user during installation. This is the short form to specify VISIBLE=Y for several components. The names of these components will show up in the Component Selection dialog (for custom installation) and in the Confirm dialog. If the component has both CHOICE and VISIBLE properties set to Y, the end user will be allowed to select or deselect this component during installation.

CISCO CONFIDENTIAL**Table 21-12 Table of Contents File – COMPONENTS Section (continued)**

DEFAULT	Comma-separated list	Comma-separated list of component tags, which are selected for installation by default. The value can be ALL. Short form to specify DEFAULT=Y for several components.
CHOICE	Comma-separated list	Comma-separated list of components for user to select. These components can be turned on or off in the Component Selection dialog. A component can be omitted during installation in one of the following cases: <ul style="list-style-type: none"> • Component has both VISIBLE and CHOICE set to Y and turned off by end user during installation. • Dependencies for component cannot be satisfied and the OPTIONAL property for that component is set to Y. (For more details, see Table 21-5 on page 21-7.) <p>Note This property provides for trivial component selection. For advance component selection, see the “Creating Advanced Component Sections” section on page 21-14. The CHOICE property is ignored if disk.toc contains one or more ADVANCED_CHOICE sections.</p>
UNINSTALLABLE	Comma-separated list	List of component tags, which can be uninstalled later. Windows only: The Uninstall <i>componentName</i> option will be created in the dialog displayed by the uninstallation process.
OVERRIDES	Comma-separated list	List of component tags, for which the <i>tag</i> sections are available in this table of contents.

Creating Overrides Sections

Each override section has a name that is a valid component tag. Parameters in this section either override those specified as component properties or provide additional properties.

The name of each section is the same as the component tag. It can be the name of the suite, installable unit, or package for which additional properties or override properties are specified in the *tag.info* or *pkginfo* file. The section name, *tag*, must be included in the OVERRIDES property in the COMPONENTS section.

Creating Advanced Component Sections

The Installation Framework supports complex component choice scenarios by specifying one or more ADVANCED_CHOICE_x sections. Each section defines a component choice scenario. There can be more than one scenario; each section describes one scenario. The section name must use this naming convention:

```
ADVANCED_CHOICE_x
where x =1, 2, ...
```

Specify these sections sequentially starting from 1. Install attempts to load these sections beginning with ADVANCED_CHOICE_1, ADVANCED_CHOICE_2, and so on, and stops when the next section is not available.

For each section, the installer checks the condition parameter (see [Table 21-13](#)). If the condition fails, the installer looks for the next section. If the condition is satisfied, the installer starts using that section and ignores the remaining sections.

CISCO CONFIDENTIAL**Table 21-13 Table of Contents File – ADVANCED COMPONENT Sections**

Name	Format	Description
ADVANCED_CHOICE_x_CONDITION	TRUE or a comma- or space-separated list of constructs <i>tag</i> [.version[-version]]. Version must be in a form minor.major.patchver, and all three of them are required or can be dropped at once with the separator. Examples: <ul style="list-style-type: none"> • cm 3.0.0 is correct • rme is correct • rme.3.1 is incorrect 	Specifies that scenario x will be used if all components specified by this parameter have been installed before. If no version is specified, then no specific version is required. If one version is specified, then the installer verifies that the version has been installed. Two version numbers specify the range of versions required. The value TRUE can be specified for the last scenario and indicates that the installer should use this scenario if conditions failed for previous scenarios.
ADVANCED_CHOICE_x_TYPE	NONE NONEXCLUSIVE EXCLUSIVE	Specifies the type of component choice. <ul style="list-style-type: none"> • NONE means there is no component choice allowed. • NONEXCLUSIVE allows the selection of one or more optional components. • EXCLUSIVE allows the selection of only one option from the list. For example: <i>ADVANCED_CHOICE_2_TYPE=NONEXCLUSIVE</i> Solaris only: If the type is NONEXCLUSIVE, the message displayed at the end of the component selection is: Select one or more items using its number separated by a comma or enter q to quit.
ADVANCED_CHOICE_x_DEFAULT	Comma- or space-separated list	If typical or custom mode is selected, this property defines the initial selection. Each choice matches the number in y in the following properties. For the EXCLUSIVE scenario, this parameter should have only one choice.
ADVANCED_CHOICE_x_y_TEXT	String, where x=1, 2, ... and y=1, 2, ...	Text shows a choice option y in scenario x.
ADVANCED_CHOICE_x_y_TAGS	Comma- or space-separated list, where x=1, 2, ... and y=1, 2, ...	Comma- or space-separated list of component tags that are selected for installation when option y of scenario x is chosen.
ADVANCED_CHOICE_x_y_DESCRIPTION	String (optional), where x=1, 2, ... and y=1, 2, ...	Description of components to be installed when the option y of scenario x is chosen.

CISCO CONFIDENTIAL**Table 21-13** Table of Contents File — **ADVANCED COMPONENT** Sections (continued)

ADVANCED_CHOICE_x_ME SG= "some message"	String	Solaris only: Displays the specified message if the ADVANCED_CHOICE_x_CONDITION is met. The message is displayed after the component selection menu.
ADVANCED_CHOICE_x_DES CRPTION= "some message"	String	Windows only: Displays the string above the component selection. Use the "\n" (two characters) to control line breaks.

Creating the PROMPT_INSTALL_TYPE Section

The PROMPT_INSTALL_TYPE section in the table of contents file (disk.toc) specifies the installation types.



Note This section is used on Solaris only. See the [“Creating the INSTALL_TYPE_SELECTION Section”](#) section on page 21-16 for similar functionality on Windows.

Table 21-14 Table of Contents File—**PROMPT_INSTALL_TYPE** Section

Name	Format	Description
PROMPT_INSTALL_TYPE_SELECTION	Comma- or space-separated list	The available installation types: <ul style="list-style-type: none"> • Typical—Installs the product using the recommended settings. • Custom—Allows the user to customize the setup options.
PROMPT_INSTALL_TYPE_SELECTION_x	String Where x is the value of the tag PROMPT_INSTALL_TYPE_SELECTION	Text that describes the installation type.
PROMPT_INSTALL_TYPE_SELECTION_Default	Typical Custom	The default installation type.

Example

```
[PROMPT_INSTALL_TYPE] PROMPT_INSTALL_TYPE_SELECTION=Typical,Custom
PROMPT_INSTALL_TYPE_SELECTION_Typical="Typical installation is
recommended for all computers." PROMPT_INSTALL_TYPE_SELECTION_Custom="Custom
installation can be
selected if you want to customize the setup options."
PROMPT_INSTALL_TYPE_SELECTION_Default=Typical
```

Creating the INSTALL_TYPE_SELECTION Section

The INSTALL_TYPE_SELECTION section specifies which installation modes (Typical or Custom) are provided.

CISCO CONFIDENTIAL

Note This section is used on Windows only. See the “[Creating the PROMPT_INSTALL_TYPE Section](#)” section on page 21-16 for similar functionality on Solaris.

Table 21-15 Table of Contents File –INSTALL_TYPE_SELECTION Section

Name	Format	Description
OPTIONS	Comma-separated list of type names.	The list of installation modes that should be provided by this installation. For example: OPTIONS=Typical,Custom Recommended: Use Typical or Custom following the guidelines in the EDCS-207385.
DEFAULT	String. Contains one of the names listed in the OPTIONS list.	The mode that should be preselected by default. Recommended default: Typical.
DESCRIPTION	String.	The description displayed above the radio boxes in the type selection dialog. Use “\n” (two characters) to control line breaks.
<type>	String. Name should match the names in the OPTIONS list.	The text to be used as a description of the setup types. For example: Typical=Use this option in most cases Custom=Installation for advanced users

Example

```
[INSTALL_TYPE_SELECTION]
OPTIONS=Typical, Custom
Typical=Typical installation. Allows you to select components. Recommended for most users.
Custom=Custom installation. Allows you to select components and customize settings. Recommended only for
advanced users.
DEFAULT=Typical
DESCRIPTION=Choose the type of Setup you prefer, then click Next.\nThis CD includes the following
components:\nCommon Services 2.2, CiscoView 5.5, Integration Utility 1.5.
```



Note Line breaks are not allowed. For example, the entire text “Typical=Typical installation. Allows you to select components. Recommended for most users.” must be specified on a single line.

Understanding the BUILD Section

The BUILD section in the table of contents file (disk.toc) contains two tags, and is generated at packaging time.



Note This section is automatically generated. The developer is not required to add anything to the disk.toc file. *Do not specify this section manually.*

CISCO CONFIDENTIAL**Table 21-16 Table of Contents File—BUILD Section**

Name	Description
BUILD_ID	The build ID of the CWCS build.
ITools_BUILD_ID	The build ID of the itools build.

Example

```
[BUILD]
BUILD_ID=SOL_CDONE2_2_20030115_0756
ITools_BUILD_ID=SOL_ITools2_2_20030115_0017
```

Creating the REQUIRED section

The REQUIRED section in the table of contents file (disk.toc) contains the list of components that must be installed before running this installation. The installer checks for these components *before* asking the user to answer other installation-specific questions. Package dependencies specified in the package properties file, on the other hand, are verified much later, after the user has finished all installation user inputs, and after running all preinstall logic.

Table 21-17 Table of Contents File—REQUIRED Section

Name	Format	Description
REQ_XXX, where XXX must match the tag and version of the component.	String	<p>The installer expects the name to contain the tag and version (or version range) separated by the colon sign (:). For example:</p> <ul style="list-style-type: none"> REQ_cdone:2.2 verifies the presence of the cdone component, version 2.2. REQ_dmgt:2.0.0-2.99.0 verifies that component dmgt is installed, and its version is in the range between 2.0 and 2.99. <p>The value should contain the message to be displayed to the user if the required component is not installed. For example, for the installer to verify if CWCS 2.2 is already installed, specify the following in disk.toc:</p> <pre>[REQUIRED] REQ_cdone:2.2=Please install CiscoWorks Common Services 3.0 before running this installation</pre> <p>The value must be specified on a single line.</p>

How the Installer Processes Properties

Build image tools convert pkgpr files into newly-named platform-specific files (*.info on Windows and *.pkginfo on Solaris). The following steps describe how the installer processes package properties:

1. Initially properties are specified in *.info or pkginfo files (see the [“Specifying Properties” section on page 21-19](#)).
2. The installer overrides component properties by those specified in the *tag* section of the Table of Contents.
3. The installer verifies the presence of the ADVANCED_CHOICE_1 section. The installer verifies all conditions in ADVANCED_CHOICE_1_CONDITION. If all conditions are matched, then scenario 1 determines options in the custom installation and default components. If ADVANCED_CHOICE_1_CONDITION is not valid, then the installer tries scenarios 2, 3, and so on until the condition is valid.

CISCO CONFIDENTIAL

4. If none of the `ADVANCED_CHOICE_x_CONDITION` conditions were met or there were no `ADVANCED_CHOICE_x` sections in the table of contents, then the component choice and default components are controlled by the sets properties `VISIBLE`, `DEFAULT`, and `CHOICE` properties in the `COMPONENTS` section of the table of contents.
5. The `VISIBLE` property lists the components which are displayed in the confirmation dialog if selected by you or by default.
6. The `UNINSTALLABLE` property in the `COMPONENTS` section specifies the list of components that will be listed for uninstallation.

Specifying Properties

Properties for packages and installable units are specified by developers and included in the protopackages. NMTG does not recommended specifying `VISIBLE`, `DEFAULT`, `CHOICE`, or `UNINSTALLABLE` properties for packages or installable units. For suites, these properties can be specified in `tag.info` files if the suite is planned for more than one release.

For each CD, the developer must create a table of contents. This file should be included in a protopackage, just like all other files. To be able to reuse protopackages in multiple images, you should include `disk.toc` in a separate protopackage, such as `<tag>.cd.tar`, where `<tag>` matches one of the packages included in the image.

This CD protopackage shall be specified as an input to the `buildImage` command, along with the protopackage containing the installation files and application protopackages.

Preparing Installation Protopackages

A protopackage contains one directory named after the tag. This directory contains the following subdirectories:

- runtime

This directory contains the directories and files to be installed on the destination server. All these files will be moved during the installation to directory specified by the user.

- On Solaris, the link `/opt/CSCOpX` will be set to the directory specified by the user and all files and directories under `tag/runtime` will be copied into `/opt/CSCOpX`.

The runtime tree on the destination server will be a result of merging of runtime subtrees of all protopackages.

- On Windows, if the user chooses the directory `C:\Program Files\My Directory`, then file `tag\runtime\objects\myAppObjects\foo` will be copied to `C:\Program Files\My Directory\objects\myAppObjects\foo`.

All files from all runtime directories will be included into `data1.cab`. Each protopackage makes a file set. The runtime directory can be empty.

- install

This directory contains the following files:

- `tag.pkgpr`. This mandatory file contains package properties. On Windows, the value of the `PKG` property must be equal to `tag`.
- `tag.rul`. This Windows-only optional file is the source code of installation hooks on Windows.

CISCO CONFIDENTIAL

- *tag.bprops* (optional). This file contains build properties each line being in the form name=value. Three values from this file are used to identify the build: PROP_ID, PROP_WHEN, and PROP_TIMESTAMP. These properties allow you to identify the build from the installed product.
- *preinstall*, *postinstall*, *preremove*, and *postremove*. These Solaris only, optional files are installation scripts.
- *disk1*. All files from this directory will be copied over to the root of the CD image. The *disk.toc* file, or table of contents, is delivered this way. Refer to the “[Creating the Table of Contents](#)” section on page 21-11 for more details.
- *cd*. All files and subdirectories from this directory will be copied to the root of the CD image. Can be used for additional files required by components that do not necessarily relate to the installation framework.
- *isupport*. The Windows-only files from this directory will be compressed into the file group Language Independent OS Independent Files of the *_user1.cab* file. During installation, all these files are decompressed into the SUPPORTDIR and can be used by installation scripts.

Related Topics

See the “[Including Files in the Protopackage](#)” section on page 21-20.

Including Files in the Protopackage

Include this file in each protopackage:

- Package properties file (*pkgpr*)—contains name value pairs for the platforms your package supports. Specifies properties for each platform. See the “[Specifying Package Properties](#)” section on page 21-6 for more information about this file.

The following files are optional:

- Build process file (*bprops*)—normally generated by the build process. Specifies when and how this particular package was built. It contains build properties lines in the form of name=value. Three values from this file are used to identify the build: PROP_ID, PROP_WHEN, and PROP_TIMESTAMP. The values of the PROP_ID and PROP_WHEN are concatenated to create the value of BUILD_ID package property. The value of the PROP_TIMESTAMP is assigned to the BUILD_TIMESTAMP package property. These properties allow you to identify the build from the installed product.
- On Windows platforms, *pkg.rul*, is an optional file, which contain Windows scripts for installation. See the “[Using the pkg.rul Installation File](#)” section on page 21-34 for more information.
- On Solaris platforms, *preinstall*, *postinstall*, *prerequisite*, *preremove*, and *postremove* files are installation scripts for UNIX and are described later in this chapter.

All files mentioned above should be included in protopackages located in the *install* subdirectory and cannot be changed.

Related Topics

See the “[Preparing Installation Protopackages](#)” section on page 21-19.

CISCO CONFIDENTIAL

Using the Installation Framework

The following topics describe the general tasks involved when using the installation framework:

- [Understanding the Common Services Upgrade](#)
- [Understanding and Implementing the casuser](#)
- [Providing Licensing Information During Installation](#)
- [Installing Database Upgrades](#)
- [Handling Patches](#)
- Application Registration with ACS during Install

Understanding the Common Services Upgrade

The CWCS team has updated the CD One 4th Edition installation and subsequent releases to disable all applications (with the exception of CiscoView) and change the ownership of the application files to the new user, casuser.

CWCS installation disables applications by unregistering daemons that do not belong to CWCS. Corresponding Windows Services will be removed. After disabling and changing file ownership, the newer version of Common Services is installed, using the installation framework upgrade procedure. As newer versions of the applications are installed, they will be able to access their data from previous versions and upgrade using the current installation framework procedure.

An additional step is required for developers updating existing code dependent on CMF 1.2 or CMF 2.2. For details on build and packaging tasks under these conditions, review the application-specific build requirements in the section “Implementing the Bin Replacement User”, which appears on page 8-36 of ENG-71441, *Bin:Bin Security Implementation Design Specification*.

Understanding and Implementing the casuser

If you have existing applications that are dependent on CMF 1.2 or earlier and must upgrade, read this topic to understand how the CWCS team has implemented security changes for owner and group.

The CWCS team has updated the CD One installation to disable all applications (with the exception of CiscoView) and change the ownership of the application files to the new user, casuser. Application developers must modify the post-install hooks and scripts to add `chown` and `chgrp` commands for each file that existed in previous CWCS or CMF releases and were not initially a part of the installation packages. This includes dynamically created files, such as data files and properties files.

To assist this process, the `resetcasuser` script in `NMSROOT/setup/support` was enhanced to allow you to randomly generate the casuser password, or enter it manually.

Related Topics

See the:

- [“Understanding the Common Services Upgrade” section on page 21-21.](#)
- [“Setting Ownership for Package Files on Solaris” section on page 21-86.](#)

CISCO CONFIDENTIAL

Providing Licensing Information During Installation

CWCS 3.0 introduced Flex LM-based licensing as described in [Chapter 34, “Using the Licensing APIs.”](#)

To provide Flex LM-based licensing information during installation:

Step 1 On Windows and Solaris, add the following entry to disk.toc:

```
PRODUCT_INFO=Product Version
```

Where:

- *Product* is the short name of the product (e.g., *cm* for Campus Manager)
- *Version* is the product version number (e.g., 4.0).

Step 2 On Windows only, add the same entry under the RELEASE section in disk.toc.

Installing Database Upgrades

This topic provides information about the structure and APIs needed to upgrade the database. This feature relies on the implementation of the `dbupdate.pl` script provided by CWCS. This feature is not required. Use this database upgrade information only if your team is using or integrating with the CWCS Server database.

NMTG developers should refer to ENG-29984 for requirements if you are bundling with RME CD (copackaging).

The following topics are covered:

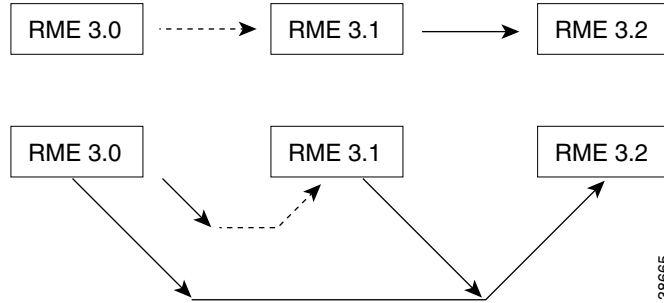
- [Upgrade Installation Paths and Strategies](#)
- [About the CWCS Upgrade Mechanism](#)
- [Adding Unauthenticated URLs](#)
- [Overriding the Dependency Handler](#)

Upgrade Installation Paths and Strategies

There are two main approaches to upgrading the database.

- An incremental approach implements a set of steps, each of them making adjustments to bring the database to the next version. These steps are executed as needed, depending on the version of a product already installed. You can implement each step as a separate script. This enables the required additional steps for each new version and only takes care of changing the database from the latest previous version to the current version.
- A one-step approach that brings the database to the current version from any previous state. You can convert all previous versions of the database to the current one.

[Figure 21-1](#) depicts both approaches for RME. Each arrow represents an upgrade script. The solid arrows represent scripts that have been implemented to upgrade Rigel releases. The dashed arrows represent scripts that will handle an upgrade to RME3.1. The dotted arrows represent scripts to handle upgrade to RME 3.2 sometime in the future.

CISCO CONFIDENTIAL**Figure 21-1 RME Upgrade Approaches**

The incremental approach assumes that the script upgrading RME3.0 to RME 3.1 handles only the upgrade from 3.0.

With the one-step approach, the script that you used to upgrade 3.0 to 3.1 needs to be replaced by the new script that is capable of upgrading from both 3.1 and 3.2. The complexity of such a script increases for each release, because the script must handle upgrades from all previous versions.

About the CWCS Upgrade Mechanism

CWCS has one mechanism to modify the database. The *dbupdate* script is integrated into the installation and enables you to register your database upgrade scripts in preinstall (Windows) or prerequisite (UNIX) hooks. Registered scripts are executed at the end of installation after all runtime files have been delivered. The database is upgraded in place.

The upgrade mechanisms described in this topic include:

- [Registration API for Upgrades](#)
- [API for Package Upgrade—CopyOut API](#)
- [Uninstall Before Upgrade](#)
- [Function Prototypes](#)

Registration API for Upgrades

The functions described here can be used in preinstall (Windows) or prerequisite (UNIX) to register the database upgrade scripts for *dbupdate*. The following table describes the functions and recommended usage.

Function	Recommended usage
RegisterDBScript	Use for drop-ins
RegisterDBScriptByVersion	Applications that use incremental approach
RegisterDBScriptByVersionEx	Applications that use a one-step approach

CISCO CONFIDENTIAL**API for Package Upgrade—CopyOut API**

To provide a useful package-based CopyOut feature, a CopyOut API has been implemented. This saves the package developers time because they do not have to use just the hook to launch the copyout script. The reason is that the “just the hook” approach in the installation framework cannot verify the total disk space required for all copyouts, because copyouts are package-independent.

You may also have to do some housekeeping work. With CopyOut API solution, multiple manifest files, different storage destinations for each package, attributes for query such as DATE_COPYOUT, PREVIOUS_VERSION, and so on can be supported. Most importantly, disk space verification is possible.

You can call this API inside the prerequisite (preinstall on Windows in pkg.rul) script. The actual copyout does not happen until all prerequisites are satisfied. The CopyOut API stores all parameters into a temporary file.

At the end of all prerequisite verification, the API verifies copyout storage requirements, plus disk space required for the installation if copyout destination happens to be in the same partition of *NMSROOT*. After the copy is done, the manifest file is copied to the storage destination location. Pkgname.properties are also created and copied to the storage destination location containing name-value pairs:

```
COPYOUT_DATE=YYYYMMDDHHMM
SOURCE_PATH=Path to copy data from specified in IF_CopyOut call
OVERWRITE_MODE=Y|N specified in IF_CopyOut call
PREV_VERSION=prevmajor.prevmajor
PREV_PATCHVER=prevpatch
NEW_VERSION=newmajor.newminor
NEW_PATCHVER=newpatch
```

Pkgname.info contains an entry pointing to this pkgname.properties. Therefore multiple entries of pkgname.properties may exist if CopyOut is executed more than once for each package using different storage locations. The property is named COPYOUT_XYZ starting with XYZ=001 and up.

For example:

```
COPYOUT_001=/opt/CiscoWorksOldData/conf/CSCOani.properties
COPYOUT_002=/opt/CiscoWorksOldData/etc/CSCOani.properties
```

Syntax	prototype IF_CopyOut(SourceDataPath, ManifestFileName, DestPath, OverwriteFileMode);		
Description	Instructs the installation framework to collect all parameters for future verification and actual copyout.		
Parameters	Field	Type	Description
	SourceDataPath	String (input)	Source path to get root data specified in manifest file. For example, <i>\$NMSROOT/conf</i>
	ManifestFileName	String (input)	File that contains the list of files to be copied out relative to SourceDataPath (ani.mfst). Files are in the same directory of the setup script (for example, disk1 location)
	DestPath	String (input)	Where to copy out. For example, <i>/opt/CW2000/CiscoWorksOldData</i>
	OverwriteFileMode	String (input)	Designates whether to overwrite existing file during the copy using “Y”/“N”.

CISCO CONFIDENTIAL**Examples**

Solaris prerequisite script:

```
#!/bin/sh

. $SETUPDIR/commonsript.sh
. $SETUPDIR/setup-lib.sh
# checking for copyout condition
# if this is an upgrade, call IF_CopyOut
if [$NEED_COPY_OUT eq 1]; then
    IF_CopyOut "$NMSROOT/conf" "ani.mfst"
              "/opt/CW2000/CiscoWorksOldData" "Y"
```

The same script on Windows:

```
Function ani_preinstall()
    NUMBER nvType, nvSize, nRc, nNeedCopyout;
    STRING bInstallingFirstTime;
Begin
    // checking for requirements
    nNeedCopyOut = 0;
    // check for upgrade case to call IF_CopyOut
    if (nNeedCopyOut = 1) then
        IF_CopyOut((NMSROOT^"conf", "ani.mfst",
                  NMSROOT^"OldData", "Y");
    endif;
end;
```

Uninstall Before Upgrade

The installation framework provides a hook to trigger package removal before the package is installed. For the package which prefers a clean installation, you can specify a combination of UNINSTALL=UPGRADE|PATCH|REINSTALL in the pkg.info(pkg.pkgpr) to make this work.

**Caution**

Data files belonging to the existing package will be deleted during the package removal. Performance is also affected. Remove the package only if there are major changes for the package.

Uninstallation for package downgrade is not supported. The action taken depends on the combination of the flags given in the UNINSTALL:

- **UPGRADE**—only uninstall the package if this is a package upgrade (CD image major.minor or is newer than the existing one. The patch version is ignored).
For example, UNINSTALL=UPGRADE uninstalls package if package is upgraded, but not during reinstall or patching.
- **PATCH**—only uninstall package if CD image major.minor version of the package is the same as existing one, and the CD image patch version is newer than the existing patch version.
For example, UNINSTALL=UPGRADE PATCH uninstalls package if it is upgraded or patched, but not during reinstall.
- **REINSTALL**—only uninstall if the CD image major.minor and patch version are the same as the existing one. Note that this only makes some difference if the package provides additional cleanup in postremove (SOL) or uninstall (Windows).
For example, UNINSTALL=UPGRADE REINSTALL uninstalls the package if the package is upgraded or reinstalled, but not during patch case.

CISCO CONFIDENTIAL**Function Prototypes**

```
RegisterDBScript(<dsn>, <script>, <args>);
RegisterDBScriptByVersion(<fromVersion>, <dsn>, <script>, <args>, <reserved>);
RegisterDBScriptByVersionEx(<fromVersion>, <dsn>, <script>, <args>, <reserved>);
```

Where:

- *dsn* is the DSN of the database;
- *script* is full installer of the script;
- *args* is the string of arguments to be passed to the script;
- *reserved* This parameter should always be empty.

Windows Example

On Windows the infsm.rul file contains:

```
function infsm_preinstall()
STRING argString, scriptName;
begin
    argString = "dsn=rme," + NMSROOT ^
        "IDS\\inventory\\desfiles\\fastswitch_wmod,CSCOinfsm";
    WriteLogFile("preinstall for infsm package");
    scriptName = NMSROOT ^ "IDS\\inventory\\scripts\\main.pl";
    RegisterDBScriptEx("rme", scriptName, argString, "");
end;
```

Solaris Example

On Solaris, the infsm.rul file contains:

```
. /opt/CSCOpX/etc/commonsript.sh
RunRequestScript CSCOinfsm
#main
echo "preinstall for CSCOinfsm";
argString="dsn=rme,/opt/CSCOpX/IDS/inventory/desfiles/fastswitch_wmod,CSCOinfsm";
scrName="/opt/CSCOpX/IDS/inventory/scripts/main.pl";
RegisterDBScriptEx "rme" $scrName $argString " ";
```

Adding Unauthenticated URLs

This API allows applications to use URLs without authentication.

Applications need some of their URL calls to be allowed without any authentication. If the system checks for authentication for every URL, some functions of the application may fail.

This API is part of the CWCS installation framework, and provides an option to use application URLs without authentication. It uses a file, `allow_files.conf`, that contains a list of URLs that should be allowed without authentication.

To make use of this API, applications should:

- Identify the list of relative URL calls to be allowed without authentication. You can find this by checking `error_log` for any HTTP forbidden errors.
- Create a file, `addURLs.txt`, in `disk1`, and include all such URL calls (that is, the URL calls to be allowed without authentication).

Example of `addURLs.txt`:

```
/CSCOnm/servlet/com.cisco.nm.cmf.servlet.webreg.csNavServlet
/help/CMF/jplug_enable.html
```

CISCO CONFIDENTIAL

```
/JSP/cm/security/AutoLogin.jsp
```

You can specify relative directory paths (with respect to web server) also in addURLs.txt. For directories, the format is as follows:

```
DIR=relative dir
```

For example, to allow the /opt/CSCOp/htdocs/swintemp directory without the authentication check, you must add the following in addURLs.txt:

```
DIR=/swintemp
```

During the installation of applications, the installation framework appends the contents of addURLs.txt to allow_files.conf.

Overriding the Dependency Handler

This feature allows you to bypass the dependency handler during installation. Use it when normal policies are believed to be inappropriate (for example, to rollback a patch).



Caution

This tool is meant to help developers get out of trouble, and is only recommended for specific situations.

Refer to the engineering spec, ENG-29971, for details about using the override.

Handling Patches

This topic provides details about how to implement patching support and supplies a checklist for developers. For examples on how to patch a CD, refer to the [“Example: Making a Patch CD”](#) section on page 21-28.

This topic covers:

- [Patch Policy](#)
- [Creating a Patch](#)

Patch Policy

Installation provides a way to prepare point patches as well as patch releases. Rollback is not supported. A typical patch contains new files that can be added to a previously installed product. *Patches must be cumulative.*

Creating a Patch

When creating a patch keep in mind that the version of a package is defined by the VERSION and PATCHVER package properties (see the [“Specifying Package Properties”](#) section on page 21-6). Developers must change the pkg.pkgpr file. Normally, the PATCHVER value should be increased for each patch.

The installer must know whether the package is a full or a sparse package:

- A **full** package contains all the runtime files that belong to the package.
- A **sparse** package contains only the files that are modified by the patch. A sparse package is indicated by adding the following property to the <tag>.pkgpr file:

```
SPARSE=YES
```

CISCO CONFIDENTIAL**Patch Mode Installation**

Patch mode enables patch delivery of one or more packages provided that a full release had been installed before. Patch mode turns off the standard dependency verification. Each package available on CD will be installed only if the same or older version of that package had previously been installed.

Packages available on the CD are skipped if the newer version of the same package has been installed or this package has not been installed. There will be no error message in this case.

The patch mode installation is enabled by using the PATCH_TAGS property in the table of content. See the [“Example: Making a Patch CD” section on page 21-28](#) for sample code to create a CWCS patch CD.

To create a patch:

Step 1 Create the table of contents for patch or patch release.

- a. Take the disk.toc of the previous release and make a copy of it.
- b. Modify the NAME and VERSTR to show the name and version of a patch release.
- c. Adjust the CHOICE, VISIBLE and DEFAULT properties to control the user interface.

The TAGS and UNINSTALLABLE properties remain unchanged if the patch release has installable units with all packages.

If release delivers only selected packages and does not have any new package, the patch mode should be used. In this case the UNINSTALLABLE property is removed and the TAG property is replaced by the PATCH_TAGS property. The PATCH_TAGS property contains the list of packages included into release.

Step 2 Fix the project file.



Note This step is required only when the NMTG build environment is used.

NMTG recommends that you create a full version on a regular basis. You can add only one CD image in the project file at this stage. Later, you can add more than one image created by one project, as needed.

Step 3 Modify sources.

Step 4 Roll up the appropriate VERSION and PATCHVER.



Note

The same uninstallation hook is responsible for removing the package both before and after applying a patch. Uninstallation of a patch is not supported. Therefore you can uninstall a package before or after a patch, but the patch cannot be reverted.

Step 5 If the package is sparse (contains only the files that are modified by the patch), add the following line to the <tag>.pkgpr file:

```
SPARSE=YES
```

Example: Making a Patch CD

The installation framework allows you to create a CD with patches. A patch CD has selected packages that can only be installed on top of the entire product. In the Patch mode, installation does not check dependencies, instead it verifies that an older version of the same package had been installed before. If

CISCO CONFIDENTIAL

an older version of a package has not been installed or the newer version of a package has been installed, that package is skipped without error messages. For instructions about making patches, see the [“Handling Patches” section on page 21-27](#).

This section provides an example of making a patch for CWCS. The first step describes creating a CD with the patch for a package. Next step describes how to make a CD containing patches for two related packages. This section uses `my_app`, `my_appdev`, and similar text to illustrate any network management package.

The following topics are covered:

- [Patching my_appdev](#)
- [Patching my_appdev and my_app](#)

Patching my_appdev

Perform the following steps to create your `my_appdev` patch.

1. [Updating the *.pkgpr](#)
2. [Building the New Protopackage](#)
3. [Making the Table of Contents File](#)
4. [Creating the CD](#)

Updating the *.pkgpr

Update your package by adding the patch version name value pair to your `pkgpr` file.

For example, if your CMF 1.0 package had version 1.0 and the `my_app.pkgpr` had the following properties:

```
NT:AIX:HPUX:SOL:
DESC=My Network Management Application
NAME=My Application
VERSION=3.0
...
```

Update your `pkgpr` file to include the `PATCHVER`:

```
NT:AIX:HPUX:SOL:
DESC=My Network Management Application
NAME=My Application
VERSION=3.0
PATCHVER=1
...
```

Building the New Protopackage

For information on protopackages, see the [“Step 3: Prepare the Make Image on Solaris” section on page 21-103](#).

Making the Table of Contents File

The following is an example of the table of contents file (see the [“Creating the Table of Contents” section on page 21-11](#)).

```
[RELEASE]
NAME=CWCS3.0 Patch CD Version 1
VERSTR=automatic build
REGISTRY_ROOT=SOFTWARE\Cisco\Resource Manager\CurrentVersion

[COMPONENTS]
PATCH_TAGS=my_appdev
```

CISCO CONFIDENTIAL

```
UNINSTALLABLE=
VISIBLE=
CHOICE=
DEFAULT=ALL
```

```
[REQUIRED]
REQ_cdone=Install Common Services 3.0 first
```

A similar file has to be created for UNIX platforms if packages listed in the PATCH_TAGS property have different tags; otherwise, the same file can be used for all platforms. The table of contents file has to be built into a protopackage and must have the name *disk.toc*. Solaris also requires the *configureMe* file:

```
nm-build3-sol251% tar -tvf cmf.patchcd.tar
tar:blocksizes = 11
drwxrwxr-x 8186/25      0 Dec 1 14:55 1999 cmf/
drwxrwxr-x 8186/25      0 Dec 1 14:55 1999 cmf/runtime/
drwxrwxr-x 8186/25      0 Dec 1 14:55 1999 cmf/disk1/
-rw-r--r-- 8186/25      199 Dec 1 14:35 1999 cmf/disk1/disk.toc
-rwxr-xr-x 8186/25      1177 Nov 24 14:27 1999 cmf/disk1/configureMe
```



Note On Windows, only the *disk.toc* file is required.

The following text is an example of part of the *configureMe* file.

```
AIX_MIN_RAM=`echo "128 * 1024" | bc`; export AIX_MIN_RAM
AIX_MIN_SWAP=`echo "$AIX_MIN_RAM * 2" | bc`; export AIX_MIN_SWAP
AIX_OS_VERSION="5.5"; export AIX_OS_VERSION
DEF_IU_ROOT="/opt/CSCOpX"; export DEF_IU_ROOT
DISKSPACE=0; export DISKSPACE
HP_MIN_RAM=`echo "256 * 1024" | bc`; export HP_MIN_RAM
HP_MIN_SWAP=`echo "1024 * 1024" | bc`; export HP_MIN_SWAP
HPX_OS_VERSION="5.5"; export HPX_OS_VERSION
INSTALL_MODE="NEW"; export INSTALL_MODE
IU_DBDIR="{IU_ROOT}/db/data"; export IU_DBDIR
IU_NAME="CSCOpX"; export IU_NAME
IU_ROOT="/opt/CSCOpX"; export IU_ROOT
LOG_NAME="ciscoininstall.log"; export LOG_NAME
PRODUCT="CSCO NM"; export PRODUCT
SERVER_EXP="[C]SCOpX"; export SERVER_EXP
SOL_MIN_RAM=`echo "128 * 1024" | bc`; export SOL_MIN_RAM
SOL_MIN_SWAP=`echo "$SOL_MIN_RAM * 1" | bc`; export SOL_MIN_SWAP
SOL_OS_VERSION="5.7 5.8"; export SOL_OS_VERSION
SWMODOPTS="-xloglevel=0"; export SWMODOPTS
SWOPTIONS="-x reinstall=true -x reinstall_files=true -x allow_multiple_versions=true -x
write_remote_files=true -x autoselect_dependencies=false -x match_target=false"; export
SWOPTIONS
UPGRADE_VERSIONS="2.* 3.0"; export UPGRADE_VERSIONS
YES_PUMP=${SETUPDIR}/install/yes.sol; export YES_PUMP
REQ_OS="SunOS"; export REQ_OS
```

Creating the CD

Use the following command to create the CD:

```
perl buildImage -r -d d:/cmfPatchImage path/is5.runtime.tar path/cmf.patchcd.tar
path/my_pkg.runtime.tar
```

For Solaris, use the *pkgtools.runtime.tar* file instead of *is5.runtime.tar*.

A patch is installed the same as a normal installation, by running the *setup.sh* script on Solaris.

CISCO CONFIDENTIAL

Patching my_appdev and my_app

You can create a patch CD containing patches for both my_appdev and my_app using the very same processes presented in the “Patching my_appdev” section on page 21-29. The only difference is that my_app must be added to the PATCH_TAGS property in the table of contents.

The command to create the CD is (all on one line):

```
perl buildImage -r -d d:/cmfPatchImage path/is5.runtime.tar
  path/cmf.patchcd.tar path/my_appdev.runtime.tar
  path/my_app.runtime.tar
```

Application Registration with ACS during Installation

When the CiscoWorks server is configured for ACS Login mode and an application /Service-Pack/IDU/drop-in is installed, install framework attempts to register all the tasks of all applications currently installed on the server with ACS. In this process, any customization of roles done in ACS is overwritten for all applications.

Starting from Common Services 3.0 Service Pack 2 (CS3.0 SP2), Common Services install framework minimizes the risk of applications getting re-registered with ACS during install/re-install/upgrade.

For Applications based on CS 3.0 SP2/Corresponding ITOOLS or higher:

If server is in ACS mode and the particular IDU/Application/SP requires new tasks to be added, appropriate warning will be provided and that Application's tasks alone will be registered.

New tasks in addition to old tasks for that applications will be registered with ACS.

Applications need to pass the following information to the intall framework, so that registration with ACS is done accordingly:

- MDC Name
- Version
- New tasks



Note

Enhancement requests have been opened against ACS. When those are addressed in ACS, the above approach will be revised.

Applications need to add a file `acsmmap.txt` under their disk1 directory. This file will be used for populating the ACS registration mapping file that CS maintains for selective registration of tasks with ACS. Install framework will also use this file to decide whether a Warning has to be displayed.

Contents of `acsmmap.txt` should be of the format:

MDC_name;app_name;app_version (with patchver)=<New tasks to be registered -Y/N>

Example: If RME 4.0.2 has new tasks to be registered with ACS, the contents will be:

```
rme;rme;4.0.2=Y
```

If an application has more than one MDC (to be registered with ACS), all of them should be mentioned in a separate line.

Example:

```
Rme1;rme;4.0.2=Y
Rme2;rme;4.0.2=N
Rme3;rme;4.0.2=Y
```

CISCO CONFIDENTIAL

The application name mentioned here should preferably map to a corresponding <appname>.info file under *NMSROOT/setup*

Example:

```
cmf, rme, cvw1
```

where there are corresponding info files. If needed, a displayable "Application name" can be picked from the INFO file. [PRODNAM attribute in the <app>.info file will be used for this purpose].

If the file *acsmmap.txt* is not available under *disk1*, the MDC name of the product will be assumed as the "application name" as well.

Applications not based on CS -ITOOls can choose to use the CLI script *AcsRegCli.pl*. The relevant documentation of such applications should be updated to recommend registering from CLI, as applicable.

AcsRegCli.pl command line script can be used to register an application without affecting the registration status of other applications. The script is located at *NMSROOT/bin*. *AcsRegCli* can be run only when the CiscoWorks server is in ACS mode.

AcsRegCli.pl has the following options:

- *listRegApp*—list the applications registered with ACS in the current CiscoWorks server.
- *listNotRegApp*—list the installed applications that are not registered with ACS in the server.
- *register <MDCName>*—register an application with ACS. <MDCName> is the name by which an application will be registered with ACS. To know this value, run *AcsRegCli.pl* with the option *-listRegApp* or *-listNotRegApp*.
- *register all*— register all the installed applications with ACS.

Refer Appendix of CS 3.0 SP2 SFS - EDCS-447583, for more details

Windows Installation Reference

This topic covers the following reference information for Windows:

- [Setting File Permissions During Installation on Windows](#)
- [Writing Windows Scripts](#)
- [Using the Windows Installation APIs](#)
- [Using Windows Build Tools](#)
- [Customizing the Installation Workflow for Windows](#)

Setting File Permissions During Installation on Windows

With the release of CWCS 3.0, the Windows "Everyone" group *does not* have read permissions for files under *NMSROOT*. It is not possible to set permissions to individual directories and files; only administrators and *casuser* have permissions to *NMSROOT* and directories or files under it.

CISCO CONFIDENTIAL

Writing Windows Scripts

Using the installation framework APIs provided with CWCS (see the [“Using the Windows Installation APIs” section on page 21-37](#)), you can write scripts that will allow you to specify any requirements and enforce constraints on a package on a Windows platform.

In addition to the basic scripts described in the following sections, you can write additional scripts to customize the logic flow of the installation (see the [“Customizing the Installation Workflow for Windows” section on page 21-76](#)).

The following topics describe how to write Windows scripts for your package:

- [Using the Windows Installation Hooks](#)
- [Using the pkg.rul Installation File](#)
- [Using Installer Global Variables](#)
- [Preloading the Global List, IAnswerFile](#)
- [Reducing Windows Installation Time](#)

Using the Windows Installation Hooks

You can use the following hooks to perform the required functions for Windows package installation:

Table 21-18 **Windows Installation Hooks**

Hook Type	Description
Preinstall	<p>The installer executes this hook before file transfer. The code in this hook should not make any changes on the target system because the user can abort installation later. Ensure that you stop execution and get the data for the components that have been installed.</p> <p>This data specifies the disk space required and questions to be asked of the installer. If the information obtained by this script is required by another script, such as preinstall, then use SetPackageProperty and GetPackageProperty APIs. If preinstall wants the installation to fail, it should call the function SetAbortFlag (see the “Using the Windows Installation APIs” section on page 21-37 for details).</p>
Postinstall	This hook is executed after file transfer. Actual configuration of component. This is the correct place to register daemons.
Uninstall	This hook is executed when the component is uninstalled right before removing files. This is a good place to unregister daemons. You can use this to clean up files and directories that were created when the product is running. For example, use this script to remove log directories and files.

Hooks are executed in the following sequence:

1. Preinstall for all components.
2. File transfer for all components.
3. Postinstall for all components.

At uninstallation time, hooks are executed in the following sequence:

1. Hooks for all components are uninstalled.
2. Files for all components removed.

It is important to understand the order in which the hooks are executed. The installer follows these rules:

- If component A depends on component B, hooks for A are executed *after* corresponding hooks for B.

CISCO CONFIDENTIAL

- The installer leaves out hooks if it decides that a particular component should not be processed. It always leaves out a component if its pending version is lower than the installed version. It can leave out a component if it is optional and not required by any other component.
- The installer executes preinstall hooks if their pending version or patchver is the same as the installed version.
- For installable units, the preinstall hook is executed *before* hooks of packages that belong to this component; postinstall is executed *after* the hooks of packages.

At uninstallation, the installer follows these rules:

- If component A depends on component B, hooks for A are executed *before* corresponding hooks for B.
- For installable units, the uninstall hook is executed *after* the uninstall hooks of components that belong to this installable unit.
- Shared packages are not removed and their uninstall hooks are not executed unless all installable units they belong to are uninstalled.

**Note**

In general, hooks should not rely on any order of execution. It is especially important to provide the same hooks for both major releases and for upgrades.

Using the *pkg.rul* Installation File

InstallShield has its own scripting language. If you use the *.rul file, the hooks are named specifically for the InstallShield. If all you plan to do with your package is to drop it in the runtime tree, you do not need to reconfigure the installation, and the *.rul file is not required. The *pkg.rul* file contains Windows scripts for installation. This is an optional file that contains InstallShield5 code for a hooks for a package or installable unit. The *pkg* must match the name of the pkgpr file described above as well as the value of PKG property.

The *pkg.rul* file begins with the line:

```
declare
followed by #define statements and function prototypes. Prototypes are needed only for additional
function, not for hooks themselves. For example:
```

```
declare
#define MY_CONSTANT5
#define ANOTHER_CONSTANT "name of file"
prototype MyFunction(String, Boolean, String);
```

After that bodies for hook functions and additional functions are included. Hook functions are InstallShield5 functions with special names.

pkg_<hook_type>,

where *pkg* is package tag and *hook_type* is one of preinstall, postinstall, or uninstall hooks. Prototypes for hook functions are generated automatically and cannot be included into the *pkg.rul* file.

For example:

```
function mypkg_preinstall()
begin
    StopService("srv");
end;

function mypkg_postinstall()
    NUMBER nRc;
```

CISCO CONFIDENTIAL

```

begin
    DmgrRegister("mydaemon", ...);
    nRc = MyFunction("mydaemon", TRUE, "parameter");
end;

function mypkg_uninstall()
begin
    DmgrUnregister("mydaemon", ...);
end;

function MyFunction(dmName, flag, msg)
begin
    ...
end;

```

Functions can take advantage of APIs and global variables. See the [“Using Installer Global Variables” section on page 21-35](#) for more details.

You can call the Core Client Registry from an installation .rul file using ccraccess.dll. See the [“Using the CCRAccess DLL” section on page 13-50](#).

Using Installer Global Variables**Caution**

If you want to create packages to be used with CWCS, *do not change* these global variable values.

The following global variables are provided by the installer and can be used directly in installation hooks. Future marketing requirements may change these values.

Variable Name	Description
<i>TOC_PRODUCT_NAME</i>	The full product name (CWCS). Use this name in user messages.
<i>TOC_SHORT_PRODUCT_NAME</i>	The short product name (also CWCS). Use this name internally to prefix other names (for example, CWCS Daemon Manager).
<i>TOC_REGISTRY_ROOT</i>	The main registry key for the product is (HKEY_LOCAL_MACHINE\SOFTWARE\Cisco\Resource Manager\CurrentVersion).
<i>TOC_VENDOR</i>	The vendor name (Cisco Systems).

To override these values, add a line to the RELEASE section of the table of contents without the TOC_ prefix. For example, the following overrides the TOC_REGISTRY_ROOT:

```

[RELEASE]
REGISTRY_ROOT=HKEY_LOCAL_MACHINE\...

```

Preloading the Global List, IAnswerFile

The installation framework automatically preloads the answer file into the global list IAnswerFile. The answer file is an ASCII file that provides the required inputs for quiet installations.

CISCO CONFIDENTIAL

Note Quiet mode is usually used internally; customers should run the installation without specifying an answer file. However, quiet mode is important for test automation and should be fully supported.

The answer file is a plain ASCII file consisting of a name=value pair on each line:

```
#--- begin answer file
#--- hash sign (#) is allowed to mark comments
adminPassword=admin
destination=d:\cscopx
systemIdentityAccountPassword=admin
#--- end of answer file
```

The CWCS installation processes the properties described in [Table 21-19](#). If your application image includes CWCS, make sure that the mandatory properties are included in the answer file. If your application image does not include CWCS, you can ignore these properties.

Assuming the above file was named `c:\answerfile`, you could call it as part of a silent installation as follows:

```
setup.exe QUIET answerfile=c:\answerfile.
```

Table 21-19 Answer File Properties

Property	Description
casuserPassword	If casuser does not exist by the time of installation, the framework generates random password for casuser. <ul style="list-style-type: none"> If the random password is successful, then no input is required. If the random password fails, installation opens a dialog requesting new password. In quiet mode, installation attempts to load the casuserPassword from the answer file. If no casuserpassword is specified in the answer file, installation attempts random password, and might fail if the random password does not pass the company policy.
destination	Allows quiet installation to install into directory other than <code>c:\Program Files\CSCOPx</code> . It is optional; if not specified, the installation is installed in <code>c:\Program Files\CSCOPx</code> .
adminPassword	Specifies the login password for the admin user, and is MANDATORY. It is only used during an original installation; reinstallation in quiet mode does not change the password.
systemIdentityAccountPassword	Password for the System Identity Account. This is mandatory.

Related Topics

For the similar procedure on Solaris, see the [“Creating the Answer File”](#) section on page 21-89.

Reducing Windows Installation Time

Developers trying to reduce total Windows installation time will want to avoid rewriting preinstall and postinstall scripts. Apart from requiring much additional developer time, this is error prone.

However, you can save substantial installation time by:

1. Editing your scripts to eliminate any global function- and variable-name conflicts.

CISCO CONFIDENTIAL

2. Once your scripts are free of these naming conflicts: Combine the existing pre- and post-installs into a single script, which can be processed automatically.

The existing build tools will attempt to do this by default (and will fail to do so if there are name conflicts), but developers have full control over whether scripts are combined or not, and which scripts are combined. To combine scripts under your control, add to `disk.toc` a `SCRIPTS` entry of the following form:

```
[SCRIPTS] combinedScript=[all_]sourceScript,...
```

Where:

- `combinedScript` is the name of the resulting
- `all_` designates that all children of the component whose script is specified in `sourceScript` need to be put into single scripts.
- `sourceScript` is the script whose children you want to combine. To specify multiple scripts, separate them with commas.

For example, to combine all CWCS scripts:

```
[SCRIPTS] cdone_a=all_cdone,all_cmfj2
```

On Windows platforms, you can also save installation time by replacing any calls to `CCRAccess.exe` with calls to `CCRAccess.dll`. See the “[Using the CCRAccess DLL](#)” section on page 13-50.

Using the Windows Installation APIs

This topic covers the APIs you can use in package-specific hooks (install shell scripts for Windows). The functions you can perform using these APIs include:

- [Accessing and Setting Package Properties to Perform Version Comparisons](#)
- [Controlling Responses to Terminated Installations](#)
- [Processing Name=Value Pairs](#)
- [Sending Informational Messages to a Log File](#)
- [Informing the Installer That a Component Requires More Space](#)
- [Registering and Unregistering CWCS Daemons](#)
- [Running Commands in a Shell](#)
- [Locating the Root Directory Path Name](#)
- [Registering and Controlling Windows Services](#)
- [Using Generic Utilities](#)
- [Managing Passwords](#)
- [Configuring Tomcat](#)
- [Controlling Reboots](#)

Accessing and Setting Package Properties to Perform Version Comparisons

Use these APIs to access and set package properties and to perform different version comparisons:

- [CompareVersion](#)
- [CompareVersionTo](#)

CISCO CONFIDENTIAL

- [GetInstalledPackageVersion](#)
- [GetPackageProperty](#)
- [IsInstalled](#)
- [LoadPackageProperties](#)
- [PROP_DIR](#)
- [RecordKeyValue](#)
- [SavePackageProperties](#)
- [SetPackageProperty](#)
- [VersionToMajorMinor](#)
- [VersionToMajorMinorPatch](#)
- [IsVersionInRange](#)
- [isPackagePending](#)

CompareVersion

```
prototype CompareVersion(STRING);
prototype CompareVersionEx(STRING, BYREF NUMBER, BYREF NUMBER,
    BYREF NUMBER);
```

Compares the version of the component being installed with the one installed before. `CompareVersionEx` also returns the version and patch level of the latest.

Arguments

Field	Type	Description
1	STRING (input)	Package/installable unit/suite tag. Contains information about package version.
2	NUMBER (output)	Major package version number.
3	NUMBER (output)	Minor package version number.
4	NUMBER (output)	Patch level version number.

Return Values

Value	Description
0	Versions are the same
<0	Pending is lower, downgrade.
>0	Upgrade.
1	Version is the same, patch level of pending component is higher.
2	Major version is the same, minor version of pending component is higher.
3	Major version of pending component is higher or it is being installed for the first time.
-1	Version is the same, patch level of pending component is lower.
-2	Major version is the same, minor version of pending component is lower.

CISCO CONFIDENTIAL

Value	Description
-3	Major version of pending component is lower.
-99	Pending version not found.

CompareVersionTo

```
prototype CompareVersionTo (STRING, NUMBER, NUMBER, NUMBER, NUMBER);
```

Compares pending or installed version of component to given values.

Arguments

Field	Type	Description
1	STRING (input)	Package/installable unit/suite tag. Contains information about package version.
2	NUMBER (input)	Installed/pending flag: <ul style="list-style-type: none"> RM_PKGPROP_PENDING—Package installed or package on CD. RM_PKGPROP_INSTALLED—Previous package installed. RM_PKGPROP_ANY—Package with latest installation version. RM_PKGPROP_CD—Package on CD.
3	NUMBER (input)	Major package version number.
4	NUMBER (input)	Minor package version number.
5	NUMBER (input)	Patch level version number.

Return Values

Value	Description
0	The same.
<0	Installed/pending/latest is lower.
>0	Installed/pending/latest is higher.
1	Version is the same, patch level of installed/pending/latest component is higher.
2	Major version is the same, minor version of installed/pending/latest component is higher.
3	Major version of installed/pending/latest component is higher.
-1	Version is the same, patch level of installed/pending/latest component is lower.
-2	Major version is the same, minor version of installed/pending/latest component is lower.
-3	Major version of installed/pending/latest component is lower.
-99	Pending version not found.

GetInstalledPackageVersion

```
prototype GetInstalledPackageVersion (STRING, BYREF STRING, BYREF NUMBER);
prototype GetPendingPackageVersion (STRING, BYREF STRING, BYREF NUMBER);
```

CISCO CONFIDENTIAL

Determines the version and patch version of component that had been installed before this installation (GetInstalledPackageVersion) or is being installed (GetPendingPackageVersion).

Arguments

Field	Type	Description
1	STRING (input)	Package/installable unit/suite tag. Contains information about package version.
2	STRING (output)	Major version and minor package version number.
3	NUMBER (output)	Patch level version number.

Return Values

Value	Description
1	Component is found.
0	Component is not found.

GetPackageProperty

prototype **GetPackageProperty**(STRING, STRING, BYREF STRING, NUMBER);
Get the value of package property.

Arguments

Field	Type	Description
1	STRING (input)	Package/installable unit/suite tag. Contains information about package version.
2	STRING (input)	Name of the property to be retrieved.
3	STRING (output)	Property value.
4	NUMBER (output)	Installed/pending flag: <ul style="list-style-type: none"> • RM_PKGPROP_PENDING—Package installed or package on CD. • RM_PKGPROP_INSTALLED—Previous package installed. • RM_PKGPROP_ANY—Package with latest installation version. • RM_PKGPROP_CD—Package on CD.

Return Values

Value	Description
0	Property extracted successfully.
-1	Property not extracted.

CISCO CONFIDENTIAL**IsInstalled**

```
prototype IsInstalled(STRING);
prototype IsInstalledEx(STRING, BYREF STRING, BYREF STRING);
```

Verifies if package, installable unit, or suite has been installed before this installation started. The `IsInstalledEx` function also returns version and information string if the component is found.

Arguments

Field	Type	Description
1	STRING (input)	Package/installable unit/suite tag. Contains information about package version.
2	STRING (output)	Major and minor version number. Uses the format <code>major.minor</code> .
3	STRING (output)	Component information as version <i>major.minor</i> installed <i>date</i>

Return Values

Value	Description
0	Component is found.
-1	Component is not found.

LoadPackageProperties

```
prototype LoadPackageProperties(STRING, LIST, NUMBER);
```

Loads property file into string list.

Arguments

Field	Type	Description
1	STRING (input)	Package/installable unit/suite tag. Contains information about package version.
2	LIST (output)	Valid string list. Functions described in Properties API can be used to obtain or modify individual properties from this list.
3	NUMBER (input)	Installed/pending flag. Start looking from pending. If not found, try the one that is already installed. <ul style="list-style-type: none"> • <code>RM_PKGPROP_PENDING</code>—Package installed or package on CD. • <code>RM_PKGPROP_INSTALLED</code>—Previous package installed. • <code>RM_PKGPROP_ANY</code>—Package with latest installation version. • <code>RM_PKGPROP_CD</code>—Package on CD.

CISCO CONFIDENTIAL**Return Values**

Value	Description
0	Loaded successfully.

PROP_DIR

STRING **PROP_DIR**; (Global variable)

Use this global directory name for temporary files. This directory is automatically removed after installation is completed. To avoid file name collisions do not use extension .info or .temp.

RecordKeyValue

```
prototype RecordKeyValue (STRING, STRING, STRING, STRING, BOOL);
prototype RestoreKeyValue (STRING, STRING, STRING, STRING, BOOL);
```

Save and restores Windows registry key values. Keys are saved permanently, and could be saved at installation time and restored at uninstallation. Each package has its own namespace to save or restore registry key values.

Arguments

Field	Type	Description
1	STRING (input)	Package tag.
2	STRING (input)	Key name.
3	STRING (input)	Key value name.
4	STRING (input)	New key value name.
5	BOOLEAN (input)	Save mode: <ul style="list-style-type: none"> For RecordKeyValue, set to TRUE to override the saved value. If last parameter is FALSE and saved value is found, value is preserved. For RestoreKey Value, controls situation when saved value is not available. If TRUE, the value specified by key name and key value name is removed.

Return Values

None

SavePackageProperties

```
prototype SaMvePackageProperties (LIST);
```

Saves preloaded properties of current component from string list.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	LIST (input)	Valid string list initialized by the LoadPackage Properties function.

Return Values

None

Notes

- This function allows saving properties only for the current package or installable unit. It does not take the package name as a parameter and does not allow modifying properties of other packages.

SetPackageProperty

```
prototype SetPackageProperty (STRING, STRING);
```

Sets the property of package or installable unit.

Arguments

Field	Type	Description
1	STRING (input)	Property name.
2	STRING (input)	Property value.

Return Values

None

VersionToMajorMinor

```
prototype VersionToMajorMinor (STRING, BYREF NUMBER, BYREF NUMBER);
```

Converts version string to numeric values.

Arguments

Field	Type	Description
1	STRING (input)	Version number in string format (major.minor).
2	NUMBER (output)	Major version as a number.
3	NUMBER (output)	Minor version as a number.

CISCO CONFIDENTIAL**Return Values**

Value	Description
0	If the string does not contain a valid version, the returned major and minor versions are both set to 0.
-1	Component is not found.

VersionToMajorMinorPatch

prototype **VersionToMajorMinorPatch** (STRING, BYREF NUMBER, BYREF NUMBER, BYREF NUMBER);

Converts patch version string to numeric values.

Arguments

Field	Type	Description
1	STRING (input)	Version number in string format (major.minor.patch).
2	NUMBER (output)	Major version as a number.
3	NUMBER (output)	Minor version as a number
4	NUMBER (output)	Patch version as a number

Return Values

Value	Description
0	If the string does not contain a valid version, the returned major and minor versions are both set to 0.
-1	Component is not found.

IsVersionInRange

prototype **IsVersionInRange** (STRING, STRING, NUMBER);

This function checks if the specified package version is within the specified range.

Arguments

Field	Type	Description
1	STRING (input)	Component tag

CISCO CONFIDENTIAL

Field	Type	Description
2	STRING (input)	The range, specified as M.m.p-M.m.p. Any lower part of a version can be omitted.
3	NUMBER (input)	Installed/pending flag: <ul style="list-style-type: none"> RM_PKGPROP_PENDING—Package installed or package on CD. RM_PKGPROP_INSTALLED—Previous package installed. RM_PKGPROP_ANY—Package with latest installation version. RM_PKGPROP_CD—Package on CD.

Return Values

Value	Description
0	The version is within limits.
-1	The version is lower than the lower limit.
1	The version is higher than the higher limit.
-99	The specified version was not found.

isPackagePending

prototype **isPackagePending** (STRING);

This function checks if the specified package is selected for installation.

Arguments

Field	Type	Description
1	STRING (input)	package tag

Return Values

Value	Description
TRUE	The specified package is selected for installation.
FALSE (0)	The specified package is <i>not</i> selected for installation.

Controlling Responses to Terminated Installations

Use the [SetAbortFlag](#) API to control a response to a terminated installation.

SetAbortFlag

prototype **SetAbortFlag** (STRING);

CISCO CONFIDENTIAL

SetAbortFlag allows your hook to terminate installation with an error message. Use in preinstalls only. Do not abort after user completes installation because that would leave the product in an unknown state. If you believe that some serious problem has occurred during postinstall, send a message (MessageBoxLog function) and proceed.

Arguments

Field	Type	Description
1	STRING (input)	Log file entry

Processing Name=Value Pairs

Use this set of APIs to process files containing name=value pairs , similar to Java property files:

- [addProperty](#)
- [addNumProperty](#)
- [loadPropertyFile](#)
- [addStringPropertyToList](#)

addProperty

```
prototype addProperty(STRING, STRING, STRING, STRING);
prototype getProperty(STRING, STRING, STRING, BYREF STRING);
```

Adds or retrieves property to and from file.

Arguments

Field	Type	Description
1	STRING (input)	Directory
2	STRING (input)	File name
3	STRING (input)	Property name
4	STRING (input or output)	Property value: <ul style="list-style-type: none"> • Input for addProperty • Output for getProperty

Return Values

- addProperty—Searches for a specified property in the file. If found, it replaces the line; otherwise a new line is appended. If this file does not exist, it is created.
- getProperty—Returns 0 if property is found successfully.

addNumProperty

```
prototype addNumProperty(STRING, STRING, STRING, NUMBER);
prototype getNumProperty(STRING, STRING, STRING, BYREF NUMBER);
```


CISCO CONFIDENTIAL

Adds or retrieves property to and from file.

Arguments

Field	Type	Description
1	STRING (input)	Directory
2	STRING (input)	File name
3	STRING (input)	Property name
4	NUMBER (input or output)	Property value: <ul style="list-style-type: none"> • Input for addNumProperty • Output for getNumProperty

Return Values

- addNumProperty—Searches for a specified property in the file. If found, it replaces the line; otherwise a new line is appended. If this file does not exist, it is created.
- GetNumProperty—Returns 0 if property is found successfully.

loadPropertyFile

```
prototype loadPropertyFile (STRING, STRING, LIST);
prototype savePropertyFile (STRING, STRING, LIST);
```

These functions allow you to preload properties from a file into a string list to speed up processing.

Arguments

Field	Type	Description
1	STRING (input)	Directory. Do not use the string list directly. Instead, use addProperty , addNumProperty , or addStringPropertyToList to set or retrieve properties.
2	STRING (input)	File name
3	LIST (input or output)	String list: <ul style="list-style-type: none"> • Input list for saveProperty • Output list for loadProperty

addStringPropertyToList

```
prototype addStringPropertyToList (LIST, STRING, STRING);
prototype addNumPropertyToList (LIST, STRING, NUMBER);
prototype getStringPropertyFromList (LIST, STRING, BYREF STRING);
prototype getNumPropertyFromList (LIST, STRING, BYREF NUMBER);
```

These functions are similar to add.../get... functions described previously.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	LIST (input)	String list
2	STRING (input)	Property name
3	STRING or NUMBER (input or output)	Property value: <ul style="list-style-type: none"> • Input for <code>addStringPropertyToList</code> and <code>addNumPropertyToList</code>. • Output for <code>getStringPropertyFromList</code> and <code>getNumPropertyFromList</code>.

Sending Informational Messages to a Log File

The following APIs enable you to send messages to a log file:

- [WriteLogFile](#)
- [AskYesNoLog](#)
- [AskYesNoLogTitle](#)
- [MessageBoxLog](#)
- [MessageBoxLogTitle](#)
- [StringListLog](#)
- [AddFileToLog](#)

The log file is for information purposes only; it is not used for uninstallation.

WriteLogFile

```
prototype WriteLogFile(STRING);
```

Writes the string into the log file. Names and locations for these log files use the convention `%SystemDrive%\CW2000_inXXX.log`, where `XXX` is equal to the log sequence (001, 002, and so on).

Arguments

Field	Type	Description
1	STRING (input)	Message to write to log file.

AskYesNoLog

```
prototype AskYesNoLog(STRING, NUMBER);
```

Displays the AskYesNo dialog and puts both the dialog message and user's reply into the log file.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	STRING (input)	Message. The same as InstallShield's AskYesNo function.
2	NUMBER (input)	Default answer.

AskYesNoLogTitle

```
prototype AskYesNoLogTitle (STRING,STRING,NUMBER);
```

Specifies the AskYesNo dialog title. This is the same as InstallShield's AskYesNo function (info, warning, error.)

Arguments

Field	Type	Description
1	STRING (input)	Dialog box title.
2	STRING (input)	Message. The same as InstallShield's AskYesNo function.
3	NUMBER (input)	Default answer

MessageBoxLog

```
prototype MessageBoxLog (STRING,NUMBER);
```

Displays the MessageBox dialog and writes a message into the log file. (The same as the InstallShield's MessageBox function.)

Arguments

Field	Type	Description
1	STRING (input)	Message. The same as InstallShield's AskYesNo function.
2	NUMBER (input)	Severity level. Predefined values: INFORMATION, WARNING, SEVERE

MessageBoxLogTitle

```
prototype MessageBoxLogTitle (STRING,STRING,NUMBER);
```

Specifies the MessageBox dialog title.

Arguments

Field	Type	Description
1	STRING (input)	Dialog box title.

CISCO CONFIDENTIAL

Field	Type	Description
2	STRING (input)	Message
3	NUMBER (input)	Severity level. Predefined values: INFORMATION, WARNING, SEVERE.

StringListLog

```
prototype StringListLog(LIST, STRING);
```

Writes all elements of string list into log file.

Arguments

Field	Type	Description
1	LIST (input)	Prefix to be added to the beginning of each element.

AddFileToLog

```
prototype AddFileToLog (STRING, STRING);
```

Copies a file into log file.

Use this function to analyze results of processes executed using the Shell Task functions (see the [“Running Commands in a Shell”](#) section on page 21-52).

Arguments

Field	Type	Description
1	STRING (input)	Path name
2	STRING (input)	Title

Informing the Installer That a Component Requires More Space

The following APIs extract the disk cluster size and inform the installer that your component requires more space than its runtime footprint:

- [GetClusterSizeEx](#)
- [needMoreSpace](#)

GetClusterSizeEx

```
prototype GetClusterSizeEx (STRING, BYREF NUMBER);
```

Determines the size of cluster for specified path name.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	STRING (input)	Valid path name
2	NUMBER (output)	Number to return cluster size

needMoreSpace

```
prototype needMoreSpace (STRING, NUMBER, NUMBER);
```

Informs installer that the component requires more space than its runtime footprint. Used only in preinstall hooks.

Arguments

Field	Type	Description
1	STRING (input)	Path name
2	NUMBER (input)	Required size in bytes
3	NUMBER (input)	Number of files

Registering and Unregistering CWCS Daemons

The following APIs enable you to register and unregister processes with the CWCS Daemon Manager:

- [DmgrRegister](#)
- [DmgrUnregister](#)

DmgrRegister

```
prototype DmgrRegister (STRING, INT, STRING, STRING, STRING);
prototype DmgrRegisterEx (STRING, INT, STRING, STRING, STRING, STRING);
prototype DmgrRegisterWR (STRING, INT, STRING, STRING, STRING, STRING, INT);
```

Registers a process with the CWCS Daemon Manager.

Arguments

Field	Type	Description
1	STRING (input)	Name
2	NUMBER (input)	Flag to run automatically or not. TRUE if daemon should be started automatically.
3	STRING (input)	Executable path name. All processes need to be registered with the default path (/opt/CSCOpX on Solaris) explicitly instead of \$NMSROOT.
4	STRING (input)	Arguments (separated by a caret).
5	STRING (input)	Daemon Manager's dependencies. A comma-separated list of other daemons on which this daemon depends.

CISCO CONFIDENTIAL

Field	Type	Description
6	STRING (input)	A flag to run in local system account. <ul style="list-style-type: none"> • 1 = Daemon should run in LocalSystem account. • 0 = Run as casuser.
7	NUMBER (input)	Maximum amount of time (in milliseconds) that Daemon Manager should wait for the process to initialize.

For cwjava daemons, use the Java versions of these APIs: `DmgrRegisterJava`, `DmgrRegisterJavaEx`, and `DmgrRegisterJavaWR`. These Java calls share the C syntax and arguments.

DmgrUnregister

```
DmgrUnregister (STRING);
```

Unregisters a daemon from the CWCS Daemon Manager. Use this function with the uninstall hook to undo registration.

Arguments

Field	Type	Description
1	String	Name of the daemon to unregister. This should match the Name parameter of the <code>DmgrRegister</code> , <code>DmgrRegisterEx</code> , or <code>DmgrRegisterWR</code> command used to register it.

Running Commands in a Shell

The following APIs enable you to run specified commands in a shell. The shell window is minimized. Control returns after command line task finishes.

- [LaunchShellAndWait](#)
- [InitBatchFile](#)

LaunchShellAndWait

```
prototype LaunchShellAndWait (STRING, STRING, BYREF STRING, BYREF STRING);
```

Runs the specified command and minimizes the shell window. It waits until the specified task is completed.

Arguments

Field	Type	Description
1	STRING (input)	Job Title. This title must be unique. It will be displayed in the task bar.
2	STRING (input)	Command line.
3	STRING (output)	File to redirect standard output. Specify as variables, not as constants.
4	STRING (output)	File to redirect standard error. Specify as variables, not as constants.

CISCO CONFIDENTIAL**Notes**

- Output files can be specified as empty, in which case standard output and standard error are not redirected. To run a batch file, use `InitBatchFile` (see the “[InitBatchFile](#)” section on page 21-53) and `RunBatchAndWait` (see the “[RunBatchAndWait](#)” section on page 21-53) .
- Either both or one of the output files can be an empty string. In this case, standard output or standard error will not be captured; it will be ignored.
- If the file name is specified, it must be a filename without a path. After this function completes, the variable of the corresponding file will contain the full path name for an output directory.

InitBatchFile

```
prototype InitBatchFile(STRING, STRING, STRING, BYREF NUMBER);
```

Initializes a batch file to execute by `RunBatchAndWait` command.

Arguments

Field	Type	Description
1	STRING (input)	Job title. This title must be unique. It will be displayed in the task bar.
2	STRING (input)	Command line.
3	STRING (input)	Temporary file name.
4	NUMBER (output)	Number to return open file handle. Used to insert more lines into batch file.

Notes

For `RunBatchAndWait` (see the “[RunBatchAndWait](#)” section on page 21-53) to work properly, the batch file must be initialized by `InitBatchFile`. Several lines can be added by `InstallShield's WriteLine` function and closed by the `CloseFile` function. Then run the `RunBatchAndWait` command using the same first and second parameters as specified for `InitBatchFile`.

RunBatchAndWait

```
prototype RunBatchAndWait(STRING, STRING, BYREF STRING, BYREF STRING);
```

Runs the batch file created by `InitBatchFile`.

Arguments

Field	Type	Description
1	STRING (input)	Job title. Must be the same title as specified for the <code>InitBatchFile</code> function.
2	STRING (input)	Temporary file name. Must be the same name as specified for the <code>InitBatchFile</code> function.
3	STRING (output)	Name of file to redirect standard output. Specify as variables, not as constants.
4	STRING (output)	Name of file to redirect standard error. Specify as variables, not as constants.

CISCO CONFIDENTIAL**Notes**

- For this function to work properly, the batch file must be initialized by InitBatchFile (see the “InitBatchFile” section on page 21-53). Several lines can be added by InstallShield's WriteLine function and closed by CloseFile function. Then run the RunBatchAndWait command using the same first and second parameters as specified for InitBatchFile.
- Either both or only the last one can be an empty string. In this case, standard output or standard error will not be captured. It will be ignored. If the filename is specified, it needs to be just a filename without a path. After this function is finished, the variable of the corresponding file will contain the full path name for an output directory.

Locating the Root Directory Path Name

The `GetRootDir` function allows you to find the path name for the root directory.

GetRootDir

```
prototype GetRootDir(STRING, BYREF STRING);
```

Find the path name for the root directory.

The installer sets actual directory path names at runtime:

1. The first time, the installer collects values of this property from all packages and installable units and asks the user to assign each one an actual path name.
2. The installer saves these values in the Windows registry.

The next time the installer runs, it gets the path names from the registry, requiring no input from the user. This allows multiple root directories, for example, one for CWCS and Essentials (*NMSROOT*) and another one for Connectivity suite (for example, *CONNROOT*).

3. The installer asks the user for path names for both directories and makes them available programmatically.

In package hooks, the value of ROOTDIR property for this package can be used directly as a global variable. If a hook needs a value of rootdir for another component, it can be obtained using the `GetRootDir` function.

Arguments

Field	Type	Description
1	STRING (input)	Root directory name.
2	STRING (output)	Path name of root directory.

Return Values

Value	Description
0	If specified root directory found.

CISCO CONFIDENTIAL**Notes**

The root directory path name for the current component is set to a global variable and can be used directly. This function is needed to obtain the path name of the root directory of another component. For example, package MYPKG has the property `ROOTDIR=MYROOTDIR`.

Example 21-1 Using GetRootDir

```
Function mypkg_postinstall()
STRING mdPropertyPath, nmsRoot;
STRING myPkgPropFileName;
begin
    // property file for this package is under its root directory and
    // MYROOTDIR can be used directly
    myPkgPropFileName = MYROOTDIR^"myPropFile.properties";
    ...
    // md.property file is somewhere under daemon manager's root - NMSROOT
    if (GetRootDir("NMSROOT", nmsRoot) = 0) then
        mdPropertyPath = nmsRoot^"subdir1\\...";
        ...
    endif
...
end;
```

Registering and Controlling Windows Services

CWCS provides a set of APIs and constants for registering and controlling Windows services. Use the following service APIs to register and control Windows services:

- [RegisterService](#)
- [UnregisterService](#)
- [StartService](#)
- [ChangeServiceStartType](#)
- [ChangeServiceAccount](#)
- [ChangeService2Casuser](#)
- [StopService](#)

Using Windows Service Constants

Whenever you use the service APIs, use the following constants to specify service types:

- `SERVICE_WIN32_OWN_PROCESS`
- `SERVICE_WIN32_SHARE_PROCESS`
- `SERVICE_INTERACTIVE_PROCESS`

You can also use the following constants to specify service start types:

- `SERVICE_BOOT_START`
- `SERVICE_SYSTEM_START`

CISCO CONFIDENTIAL

- SERVICE_AUTO_START
- SERVICE_DEMAND_START
- SERVICE_DISABLED

RegisterService

prototype **RegisterService** (STRING, STRING, STRING, LONG, LONG);

Registers new Windows Services.

Arguments

Field	Type	Description
1	STRING (input)	Service name
2	STRING (input)	Service display name
3	STRING (input)	Path name of executable
4	LONG (input)	Service type. Specify a service type constant (see the “Using Windows Service Constants” section on page 21-55).
5	LONG (input)	Service start type. Specify a service start type constant (see the “Using Windows Service Constants” section on page 21-55).

UnregisterService

prototype **UnregisterService** (STRING);

Unregisters Windows services.

Arguments

Field	Type	Description
1	STRING (input)	Service name

StartService

prototype **StartService** (STRING);

Starts Windows services immediately.

Arguments

Field	Type	Description
1	STRING (input)	Service name

ChangeServiceStartType

prototype **ChangeServiceStartType** (STRING, LONG);

Changes service start type.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	STRING (input)	Service name
2	LONG (input)	Service start type. Specify a service start type constant (see the “Using Windows Service Constants” section on page 21-55).

ChangeServiceAccount

```
prototype ChangeServiceAccount (szServiceName, accountName,
    accountPassword);
```

Changes the service account. Use this function after the service is created (see the [“RegisterService”](#) section on page 21-56).

Arguments

Field	Description
szServiceName	The name of the service.
accountName	The account for the service to run as. To modify the service to run as LocalSystem account, the value should be “.LocalSystem” (without quotes, and remember to escape the backslash).
accountPassword	The password for the account. Leave this empty if the account is LocalSystem.

Related Topics

See the [“ChangeService2Casuser”](#) section on page 21-57.

ChangeService2Casuser

```
prototype ChangeService2Casuser (STRING, POINTER);
```

This function reconfigures a service to run as casuser. This relieves the developer from the need to fetch the value of the casuser password, which is maintained by the Daemon Manager. This function can only be used after the Daemon Manager has been installed.

Arguments

Field	Type	Description
1	String	The name of the service.
2	Pointer	The pointer to the string that contains the password. Specify NULL to use the password stored by the Daemon Manager.

Return Values

0 if successful.

CISCO CONFIDENTIAL**Example**

This function is implemented in `secure.dll`, which is a part of the installation framework. Therefore, a call to this function must be framed with the `UseDll/UnUseDll` calls. For example:

```
if (UseDLL(SUPPORTDIR ^ "secure.dll") != 0) then
    MessageBoxLog("Cannot load secure.dll", SEVERE);
endif;
ChangeService2Casuser("myService", NULL);
UnUseDll(SUPPORTDIR ^ "secure.dll");
```

StopService

```
prototype StopService(STRING, NUMBER);
```

Stops service.

Arguments

Field	Type	Description
1	STRING (input)	Service name
2	NUMBER (input)	Stop mode. Options: <ul style="list-style-type: none"> RM_STOPSERV_IMMEDIATELY RM_STOPSERV_DELAYED

Notes

- In preinstalls, use `RM_STOPSERV_DELAYED`. The installer does not stop services immediately; instead, it collects the list of services to stop.
- After all preinstalls are executed, the installer asks the user for confirmation and stops all requested services at once. In addition, the installer stops all services that were requested to be stopped.
- In all hooks other than preinstall, use the `RM_STOPSERV_IMMEDIATELY` mode.

Using Generic Utilities

The generic installation framework utilities contain functions for string list processing, path processing, deleting files, displaying system error messages, and obtaining DNS names:

- [EmptyList](#)
- [InvertList](#)
- [GetStringFromList](#)
- [ListFindSubstring](#)
- [AddDirToPath](#)
- [DeleteFromPath](#)
- [pathToFS](#)
- [ShowLastSysError](#)
- [SureDeleteFile](#)
- [GetHostName](#)

CISCO CONFIDENTIAL

- [ModifySubParameter](#)

EmptyList

prototype **EmptyList**(LIST);

Removes all elements from the list.

Arguments

Field	Type	Description
1	LIST (input and output)	Name of valid string list. Input list will be modified.

InvertList

prototype **InvertList**(LIST);

Changes the order of elements in a string list.

Arguments

Field	Type	Description
1	LIST (input and output)	Name of valid string list. Input list will be modified.

GetStringFromList

prototype **GetStringFromList**(LIST, NUMBER, BYREF STRING);

Retrieves an element from a string list by number. For example, it will return the third string in a list.

Arguments

Field	Type	Description
1	LIST (input)	Name of valid string list
2	NUMBER (input)	Index
3	STRING (output)	String returned by the function. It is “ ” if the requested string is not available.

ListFindSubstring

prototype **ListFindSubstring**(LIST, STRING, NUMBER, BYREF STRING);

Finds an element in a string list, which contains given string starting from the current element.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	LIST (input)	Name of valid string list.
2	STRING (input)	String to search for.
3	NUMBER (input)	Offset. -1 stands for any offset.
4	STRING (output)	String returned by the function. The same as ListGetNextString. It is “ ” if the requested string is not available.

Notes

- InstallShield’s ListFindString searches for an element that is exactly similar—it does not allow searching for a substring.
- The third parameter allows you to specify an offset. For example, to find an element that begins with “ABC”, call ListFindSubstring(listID, “ABC”, 0, svValue).

AddDirToPath

prototype **AddDirToPath**(STRING);

Adds a specified directory to the path.

Arguments

Field	Type	Description
1	STRING (input)	Path name of directory to be added

DeleteFromPath

prototype **DeleteFromPath**(STRING);

Deletes path name from the path.

Arguments

Field	Type	Description
1	STRING (input)	Path name to be removed.

pathToFS

prototype **pathToFS**(BYREF STRING);
 prototype **pathToBackSlash**(BYREF STRING);

Converts backslashes in path name into forward slashes and vice versa.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	STRING (input)	String that contains path name. After execution, the modified path name.

ShowLastSysError

```
prototype ShowLastSysError (STRING, LONG);
```

Display system error message in dialog box.

Arguments

Field	Type	Description
1	STRING (input)	Title of dialog box
2	LONG (input)	Error code

Example

```
ShowLastSysError ("OpenSCManager failure",GetLastError());
```

SureDeleteFile

```
prototype SureDeleteFile (STRING);
```

Removes specified file.

Arguments

Field	Type	Description
1	STRING (input)	Installer of a file to remove. Processes both short and long file names

GetHostName

```
prototype GetHostName (BYREF STRING);
```

Retrieves the DNS host name.

Arguments

Field	Type	Description
1	STRING (output)	String variable to receive host name.

ModifySubParameter

```
prototype ModifySubParameter (BYREF STRING, STRING, STRING, BOOL, STRING, STRING);
```

Modifies the value of a subparameter in a command line or removes it.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
1	STRING (output)	String to modify.
2	STRING (input)	Parameter whose value you want to modify.
3	STRING (input)	New subparameter or subparameter to remove.
4	BOOL (input)	TRUE to add or replace subparameter; FALSE to remove subparameter.
5	STRING (input)	Delimiters for parameters
6	STRING (input)	Delimiters for subparameters

Notes

For example, you can modify the line:

```
myExe.exe -d p1,p2,p3 -p jhgjhg -t
```

to look like

```
myExe.exe -d p1,p2,p3,newP -p jhgjhg -t
```

by calling

```
ModifySubParameter(myStr, "-d", "newP", TRUE, " ", " ,")
```

where `myStr` has a string to modify.

Managing Passwords

In this release, there are two sets of functions that manage passwords:

- The functions `AskPass` and `GetPass` provide CMF 2.1-style password management, where the user can modify the value of the default password.
- In CWCS 2.2, the functions `SetPassEx`, `GetPassEx`, and `AskPassEx` facilitate adding password-related dialogs as a part of the installation user interface.

This section contains the password management APIs:

- [AskPass](#)
- [GetPass](#)
- [AskPassEx](#), [AskPassExTitle](#)
- [GetPassEx](#)
- [SetPassEx](#)
- [ChangeDbPasswd.pl](#)

AskPass

AskPass (`Prompt_String`, `Temp_File_Name`, `DbPass`, `Password`)

The `AskPass` API:

1. Prompts for confirmation to change the password. Enter Yes or No.
 - If you enter Yes, `AskPass` prompts for the new password.

CISCO CONFIDENTIAL

- Enter the new password, confirm it, and click **Next**.
- 2. Validates the password and stores it in a temporary file.
- 3. Encrypts the file using the cipher command.

Arguments

Field	Type	Description
Prompt_String	String	The confirmation prompt.
Temp_File_Name	String	Name of the temporary file where the password is stored.
DbPass	Integer	Used to validate the password. When the value of this argument is <i>1</i> , the password is validated according to database rules. For example: <ul style="list-style-type: none"> • The password should not start with a number. • The password should not contain any special characters and Password length should not be more than 15 characters. When set to values other than 1, no validation is done.
Password	String	Displays the user-entered or randomly-generated password.

Notes

This API is called in the preinstall function of the Db.rul.

Example

```
AskPass("Enter Common Services Database Password",
        "Changing Database Password",
        "enpass_cmf", 1);
```

GetPass

```
GetPass(Temp_File_Name, Password)
```

Use this function to retrieve the value saved by the [AskPass](#) function. The GetPass API:

1. Retrieves the password from the temporary file.
 - GetPass accepts the file name and a string variable as the parameter.
2. Decrypts the temporary file and returns the password in the variable.
3. Deletes the file once the password is read.
 - GetPass returns 0 if it succeeds in getting the password, and 1 if it fails.

Arguments

Field	Type	Description
Temp_File_Name	String	Name of the file where the password is stored.
Password	String	Variable where the password will be returned

Example

```
if (GetPass("casuser_pass",passwd) = 0) then
...

```

CISCO CONFIDENTIAL

In this example, the API reads the password from the file `casuser_pass`. The variable `passwd` contains the password.

AskPassEx, AskPassExTitle

```
prototype AskPassEx (szDescription, szComment, dialogId, szDLL,
    fileName, mode, lLabels, nIdStartPwd);
prototype AskPassExTitle (szDescription, szComment, dialogId, szDLL,
    fileName, mode, lLabels, nIdStartPwd, szTitle);
```

These functions prompt the user to specify a new password or accept the default or existing password. The value of the password is stored at a temporary location in encrypted format and can be retrieved by the `GetPassEx` API (see the “[GetPassEx](#)” section on page 21-65). Use this function in the custom panel. For details and a sample, see the “[Customizing the Installation Workflow for Windows](#)” section on page 21-76.

Arguments

Argument	Description
<code>szDescription</code>	The text of description, displayed above the input fields.
<code>szComment</code>	The text of the comment, displayed below the input fields.
<code>dialogId</code>	ID of the dialog template. Currently, the installation framework contains two templates: <ul style="list-style-type: none"> 13039 (with input fields for one password with confirmation) 13034 (with input fields for two passwords with confirmations).
<code>szDLL</code>	The name of the DLL that contains the dialog templates. Use an empty string for templates supported by the installation framework.
<code>fileName</code>	The name of temporary file that stores the values for the duration of installation. Do not use path names.
<code>mode</code>	Password verification mask. Can be a combination of the following: <ul style="list-style-type: none"> <code>PASS_EX_ANY</code>—Any value is allowed (should not be combined with any other value). <code>PASS_EX_MAX_LENGTH</code>—The maximum length of the password. <code>PASS_EX_NONEMPTY</code>—The password must be not empty (should not be combined with <code>PASS_EX_EMPTY_ALLOWED</code>). <code>PASS_EX_EMPTY_ALLOWED</code>—The empty password is allowed (should not be combined with <code>PASS_EX_NONEMPTY</code>). <code>PASS_EX_ALPHANUM</code>—The password can only contain digits and alphabetic characters. <code>PASS_EX_FIRST_ALPHA</code>—The first character must be alphabetic. <code>PASS_EX_MIN_LENGTH</code>—The minimum length of the password. The example illustrates some of these options.
<code>lLabels</code>	List of the labels of input fields. Should contain labels for fields, followed by the labels of confirmation fields (optional). By default, the confirmation fields are labelled “Confirm”/.

CISCO CONFIDENTIAL

Argument	Description
nIdStartPwd	ID of the first input field. For the dialogs supported by the framework, use 0.
szTitle	(In AskPassExTitle only)—The title of the dialog. The dialog displayed by AskPassEx uses the title “Change Password”

Return Codes

NEXT or BACK, depending on whether the user clicked the Next or Back buttons.

Example

The following code fragment displays a dialog that lets the user specify values for two passwords (with confirmation for each of them), and stores the result in the admin_pass file in a temporary location. Both passwords cannot be empty, must be at least 5 characters long, contain only alphanumeric characters, and the first character must be alphabetical.

```
lLabels = ListCreate(STRINGLIST);
ListAddString(lLabels, "User admin Password", AFTER);
ListAddString(lLabels, "User guest Password", AFTER);
nRc = AskPassExTitle("You may change the password of admin and guest users.\n" +
    "Leave fields empty to keep existing passwords.",
    "Password must begin with an alphabetic character and should be "+
    "at least 5 characters long.",
    13034,"13034","", "admin_pass",
    PASS_EX_NONEMPTY + PASS_EX_ALPHANUM +
    PASS_EX_FIRST_ALPHA + PASS_EX_MIN_LENGTH * 5,
    lLabels, 0, "Change Admin and Guest Password");
ListDestroy(lLabels);
```

Related Topics

See the:

- [“GetPassEx” section on page 21-65.](#)
- [“Customizing the Installation Workflow for Windows” section on page 21-76.](#)

GetPassEx

```
prototype GetPassEx (szFileName, svField, nField);
```

This API retrieves the values stored by AskPassEx or SetPassEx.

Arguments

Name	Description
fileName	File name. Must match the fileName parameter of the AskPassEx or SetPassEx functions.
svField	The variable that receives the value.
nField	The field number (counting from 0).

CISCO CONFIDENTIAL**Return Values**

Value	Meaning
0	Successful
1	Not successful

SetPassEx

```
prototype SetPassEx (IData, fileName);
```

Stores values in the temporary location in an encrypted format. This is suitable to pass values from early stages of installation to later stages. Encryption will only work on a Windows operating system that supports encryption.

Arguments

Name	Description
IData	List of values to be preserved.
fileName	The name of temporary file that stores the values for the duration of the installation. Do not use path names.

Return Values

Value	Meaning
0	Successful
1	Unsuccessful

Notes

Subsequent calls with the same fileName parameter will replace the previously stored values.

ChangeDbPasswd.pl

```
$NMSROOT\bin\perl.exe
$NMSROOT\objects\db\conf\ChangeDbPasswd.pl DSN New_Password
```

This API changes the password in the database and property files. It accepts the DSN name and password as the parameters and changes the password.



Note This API is normally called in the postinstall function of the DB package.

The ChangeDbPasswd.pl API:

- Checks whether the DSN is valid.
- (If the DSN is valid) checks whether the database is enabled,
- (If the database is enabled), changes the password in the database, odbcmpl, .odbc.ini, DBServer.properties and the property file specified in .odbc.mpl.
- (If the database is not enabled), changes the password in the database and odbcmpl.

CISCO CONFIDENTIAL**Arguments**

Field	Type	Description
DSN	string	Database Name
Password	string	New Password

Return Values

Value	Description
0	Password was changed successfully.
>0	Password change failed.

Example

```
$NMSROOT\bin\perl.exe
$NMSROOT\objects\db\conf\ChangeDbPasswd.pl cmf cisco
```

Configuring Tomcat

The Tomcat configuration APIs include:

- [ModifyFolderXML](#)
- [UpdateTomcat](#)

ModifyFolderXML

ModifyFolderXML(fileName, replacedString, contextPath, finalString)

Configures the application Desktop XML file to integrate it with CWCS.

Arguments

Field	Description
fileName	Required. The folder file you want to integrate.
replacedString	Required. The placeholder to be replaced with CWCS URL information.
contextPath	Required. The URL pattern that identifies your servlet context.
finalString	Optional. The string to use as a replacement within your file. The default string provided by CWCS is based on data from CCR regarding the server name, protocol, and port number.

Notes

This method delegates its work to a MICE installation utility class. The actual command line that will be run looks like this:

```
<path to java>/java com.cisco.core.mice.util.install.FolderXMLModifier <Absolute Path to XML> <String to replace> <context url pattern> <String to prepend>
```

CISCO CONFIDENTIAL**Example**

In this example,

```
java com.cisco.core.mice.util.install.FolderXMLModifier
  C:/MyApp.xml REPLACE_THIS_STRING /myApp
```

This command will replace all instances of the string REPLACE_THIS_STRING in the drawer file with the default prepend value from CCR, which consists of the URL values to talk to Common Services and initialize the session data between CiscoWorks applications and Common Services.

The ModifyFolderXML utility can transform this line fragment in the myApp.xml file from this:

```
<item NAME="Launch myApp"
  HREF="REPLACE_THIS_STRING/myapp?command=firstPage">
```

to this (all one line):

```
<item NAME="Launch myApp"
  HREF="/CSCOnm/servlet/com.cisco.core.mice.util.cmf.
  CMFLIaisionServlet?command=initializeAndValidate&
  url=%2Fmyapp%3Fcommand%3DfirstPage&
  context=/myapp&port=443">
```

This method:

- Sets the URL up to bounce to the CMFLiaisonServlet on the Tomcat servlet engine.
- Passes information to that servlet about what port the Common Services Tomcat engine is running on, and what servlet context the eventual URL target is going for.
- URLEncodes the original URL to preserve whatever data or parameters an application is attaching to their requested URL.

UpdateTomcat

```
UpdateTomcat (aUrlPattern, aRelativePath, aMapping, appid)
UninstallUpdateTomcat (aUrlPattern, aRelativePath, aMapping, appid)
```

Call the UpdateTomcat method when you are finished populating CCR (the CWCS Client Registrar).

This method:

Handles the configuration of Apache and Tomcat.

- Populates CCR with the information required by the application servlet context to operate within the core multi-context environment.
- Adds the JkMount directive to httpd.conf.
- Adds the <Context> node into the apps-ordered.xml file of Tomcat.
- Creates a custom ContextInfo entry in CCR.

The UninstallUpdateTomcat method cleans up data related to the application.

Both methods perform some configuration, but delegate most of the work to the Java utility class TomcatServiceUpdate. This class is called from the command line within the installation framework.

CISCO CONFIDENTIAL**Arguments**

Field	Description
aUrlPattern	The application's context path.
aRelativePath	The application's docbase relative to Tomcat.
aMapping	The servlet mapping of the application's CWCS liaison servlet.
appid	The ID that identifies this application to CAM.

Notes

All arguments must be defined for full integration and interaction to occur with CWCS.

Example

This is a custom ContextInfo entry in CCR:

```
<Custom22>
  <Name>ContextInfo</Name>
  <Location>/testLiaison</Location>
  <Data>/myApp</Data>
  <appid>myApplication</appid>
  <ReferenceCount>1</ReferenceCount>
  <References>
    <Core />
  </References>
</Custom22>
```

Controlling Reboots

The [SetRebootFlag](#) function allows you to request a reboot at the end of an install or uninstall.

SetRebootFlag

```
SetRebootFlag()
```

Causes the framework to request a system reboot from the user at the end of an installation or uninstallation.

Arguments

None.

Notes

This method tells the installation framework that the system should be rebooted once the installation or uninstallation is finished. The framework will open a dialog requesting the user to confirm the reboot. The user can choose to either reboot at the time of the prompt or to “reboot later”.

You can use SetRebootFlag in a Preinstall or Postinstall to request reboot at the end of installation, or in an Uninstall to reboot after uninstallation.

You can also request reboot at the end of uninstallation by adding UNINSTALL_REBOOT=Y to the package properties file (see [Table 21-5 on page 21-7](#)).

CISCO CONFIDENTIAL**Example 21-2 Using GetRootDir**

```

Function mypkg_postinstall()
STRING mdPropertyPath, nmsRoot;
STRING myPkgPropFileName;
begin
    // property file for this package is under its root directory and
    // MYROOTDIR can be used directly
    myPkgPropFileName = MYROOTDIR^"myPropFile.properties";
    ...
    // md.property file is somewhere under daemon manager's root - NMSROOT
    if (GetRootDir("NMSROOT", nmsRoot) = 0) then
        mdPropertyPath = nmsRoot^"subdir1\...\.";
        ...
    endif
    ...
end;

```

Using Windows Build Tools

This topic provides instructions for building an image from protopackages using the installation framework. The main steps are:

- [Step 1: Install Third-Party Tools for Windows](#)
- [Step 2: Install the Framework on Windows Platforms](#)
- [Step 3: Prepare the Make Image on Windows Platforms](#)

The following topics provide additional guidelines and examples:

- [Debugging on Windows Platforms](#)
- [Example: Using Windows Build Tools](#)

For examples showing how to add your application to the CWCS image, see the “[Solaris Getting Started Example](#)” section on page 21-106.

Step 1: Install Third-Party Tools for Windows

Install the Windows third-party tools listed in the “[Third-Party Tools for Installation Framework](#)” section on page 21-4.

Step 2: Install the Framework on Windows Platforms

To install the framework, copy the following files to your Windows hard drive:

- `buildImage`: This main script creates the CD image from protopackages.
- `verifyImage`: This script verifies the structure of the protopackages.
- `is5.runtime.tar`: This protopackage contains the Windows installation framework.

CISCO CONFIDENTIAL**Step 3: Prepare the Make Image on Windows Platforms**

Follow these steps to prepare the make image on Windows platforms:

Step 1 Verify that MKS is in your path, by entering the following commands:

```
which sh
which perl
which find
```

The returned path names for shell, perl, and find should refer to the MKS tools. If this is not true, modify the path to include MKS binaries.

Step 2 If InstallShield or PackageForTheWeb are installed in non-default directories, set the following environment variables:

MK_IS55=full path where InstallShield is installed

MK_PFTW=full path where PackageForTheWeb is installed

For both environment variables use backslashes with short path name. Spaces are not allowed. For example

```
set MK_IS55=d:\progra~1\instal~1\instal~1.5pr CORRECT
set MK_PFTW=d:\progra~1\instal~1\packag~1 CORRECT
set MK_IS55= D:\Program Files\InstallShield\InstallShield 5.5 Professional Edtn INCORRECT
```



Note The short path name can be specific to your system. To determine the short path name, use the command **dir /x**.

Step 3 Run the buildImage command:

```
perl buildimage -r -d image_path rtpath/is5.runtime.tar pkgpath/myPkg.runtime.tar
protopackage...
```

Where:

- *-r* is an option to refresh the *image_path* area by first removing all directories from *image_path*\extract and then extracting all protopackages.
- *image_path* is the full path name of a directory where the image will be created. A lot of free space is required on this drive, at least three times the size of the runtime. You must specify the path starting from the drive letter, with forward slashes, no spaces allowed.
- *rtpath* and *pkgpath* are the full path names of the directories where the is5.runtime.tar and you package are stored, respectively
- *protopackage* should be specified with full path names with forward slashes, no spaces allowed. For the complete list of protopackages, see the sample scripts below.

This command will create the following directories under *image_path*:

- *extract*: This contains all files extracted from protopackages.
- *temp*: This contains temporary files required to arrange an image.
- *disk1*: This contains the installable image.
- *ReleaseName.exe*: This is the self-expanding installable image. *ReleaseName* will be similar to the name of the release as specified in the table of contents.

CISCO CONFIDENTIAL**Debugging on Windows Platforms**

After the `buildImage` script has created the `extract` directory, you do not have to extract protopackages again. Instead, you can change into the `extract\is5\install` directory and run the `runme.sh` script:

```
cd myImage_path\extract\is5\install
sh runme.sh > myImage_path\my.log 2>&1
```

You can apply changes directly to the files under the `extract` directory. Do not forget to put changes back into corresponding protopackages.

InstallShield's debugging information is available under `image_path\temp\isproject\Script Files*.dbg`. To run installation hooks in the debugger, copy the corresponding `.dbg` file to the `disk1` directory with the image and run installation from the command line with the `DEBUGHOOKS` parameter. For example:

```
cd myImage_path\disk1
copy ..\temp\isproject\Script files\tag.dbg
setup DEBUGHOOKS
```

Where `tag` is the tag of the protopackage you want to debug.

Installation will run `preinstall` and `postinstall` scripts in the debugger. It can also open message boxes complaining about `.dbg` files not being available for other protopackages. Click OK and proceed with the installation.

The installation framework contains a batch file, `rul.cmd`, that simplifies the development process on Windows. This file:

- Is located in the `is5.runtime.tar` in the `is5\install` directory. The `buildImage` command creates the `extract` directory, as well as the installable image.
- Lets you modify the hooks directly in the `extract` area, and then immediately recompile them into the image.
- Requires the `IMAGE` environment variable. It should be set to the pathname of the directory that contains the `extract` area. For example, if the `extract` area was created in the `d:\imagePath` directory, specify the `IMAGE` environment variable like this:

```
set IMAGE=d:\imagePath
```

To recompile the code for the hook of `myApp` package, modify the code in the `d:\imagePath\extract\myApp\install\myApp.rul` file, then run the following command:

```
d:\imagePath\extract\is5\install\rul.cmd myApp
```

This command recompiles the code directly into `d:\imagePath\disk1` and copies the debugging information (`myApp.dbg` file).

Example: Using Windows Build Tools

The following topic provides examples on how to create an installable CWCS CD image with a customer application. In these examples, we assume the following:

- The `myapp` name refers to an application called *my application*. The application name is truncated to meet the Solaris requirement for five-letter package names (`CSCOxxxx`).
- Our sample application is in two tar-files, `myapp.cd.tar` and `myapp.runtime.tar`, both in the current directory. These files are copied automatically by the installation script to the following target directories: `myapp.cd.tar` into `myapp\disk1`, and `myapp.runtime.tar` into `myapp\install` and `myapp\runtime` structures.

CISCO CONFIDENTIAL

The directory myapp\disk1 contains the following disk.toc file describing our sample application's table of contents.

```
[RELEASE]
NAME=CWCS with Test Application
VERSTR=2.0
REGISTRY_ROOT=SOFTWARE\Cisco\Resource Manager\CurrentVersion

[COMPONENTS]
TAGS=cwcs cmfwd cmfj2 myapp
UNINSTALLABLE=cmfwd myapp
VISIBLE=cwcs cmfwd myapp
CHOICE=cwcs cmfwd myapp
DEFAULT=ALL

[ADVANCED_CHOICE_1]
ADVANCED_CHOICE_1_CONDITION=cmfwd.1.0.0-1.9.99
ADVANCED_CHOICE_1_TYPE=EXCLUSIVE
ADVANCED_CHOICE_1_DEFAULT=4
ADVANCED_CHOICE_1_1_TEXT=CiscoWorks Common Services (CWCS) Base Desktop
ADVANCED_CHOICE_1_1_TAGS=cmfwd
ADVANCED_CHOICE_1_2_TEXT=CWCS (including Base Desktop)
ADVANCED_CHOICE_1_2_TAGS=cwcs
ADVANCED_CHOICE_1_3_TEXT=myapp Application
ADVANCED_CHOICE_1_3_TAGS=myapp
ADVANCED_CHOICE_1_4_TEXT=myapp Application and CWCS
ADVANCED_CHOICE_1_4_TAGS=cwcs cmfwd myapp

[ADVANCED_CHOICE_2]
ADVANCED_CHOICE_2_CONDITION=TRUE
ADVANCED_CHOICE_2_TYPE=EXCLUSIVE
ADVANCED_CHOICE_2_DEFAULT=3
ADVANCED_CHOICE_2_1_TEXT=CiscoWorks Common Services (CWCS) Base Desktop
ADVANCED_CHOICE_2_1_TAGS=cmfwd
ADVANCED_CHOICE_2_2_TEXT=CWCS (including Base Desktop)
ADVANCED_CHOICE_2_2_TAGS=cwcs
ADVANCED_CHOICE_2_3_TEXT=myapp Application and CWCS
ADVANCED_CHOICE_2_3_TAGS=cwcs cmfwd myapp
```

For this toc file you need the following:

- REGISTRY_ROOT provides the key name for component information. The value has been set for the CWCS release and has to be the same to keep the compatibility with the CWCS.



Note The NAME parameter in the [RELEASE] section is used to make a name of an executable. All spaces are replaced by underscores, but parenthesis, comma slashes, and other characters are not allowed in filenames.

The ADVANCED_CHOICE section determines what is the correct scenario depending on what is already installed on the target machine. In the case where the target machine has a previous version of CWCS Desktop already installed, the customer has the following options:

- Upgrade to a new version of CWCS Desktop
- Install CWCS 3.0
- Install the application myapp on top of CWCS Base Desktop
- Install his application bundled with CMF 2.2.

In the case of a fresh installation, you don't have the option of a separate myapp installation. So in this case there are only the remaining three options mentioned in ADVANCED_CHOICE_2 section.

CISCO CONFIDENTIAL

- The directory `myapp\install` contains the `myapp.bprops` and `myapp.pkgpr` files. The file `myapp.bprops` provides build identification. For example:

```
D:\myapp\install>cat myapp.bprops
PROP_ID = build_test
PROP_TIMESTAMP = 939800833
```

The `myapp.pkgpr` file contains package properties that specify the name, version, and dependencies. For example:

```
D:\myapp\install>cat myapp.pkgpr

NT:AIX:HPUX:SOL:
NAME=Sample Package
DESC=This is an example of installation framework
VERSION=5.0
PATCHVER=0
DEPENDS=cmfwd

SOL:
PKG=CSComyapp

NT:
PKG=myapp
```

The directory `myapp\runtime` contains the `cgi-bin` and `htdocs` subdirectories, corresponding to the structure of similar directories of the CiscoWorks-product to be used by the web-server. Two Perl scripts, `cgi-bin\myappmyappcgi.pl` and `htdocs\myapp\myappframe.html` file. Each developer must replace these files with their application files. Two files, `htdocs\Xml\System\maintree\myapp.xml` and `htdocs\Xml\System\maintree\myappcgi.xml`, are for linking our html-and Perl-files to the appropriate tree structure of the CiscoWorks Home Page. Each developer must swap their xml-files in their place. For the details on integrating your application with CiscoWorks, refer to the [“Integrating Your Application with CWHP” section on page 7-6](#).

- The protopackages for CMF 2.0 are in the `\...\protopackages` directory of the main SDK kit.
- `is5.runtime.tar` file is in `\...\protopackages` directory of the main SDK kit.
- `BuildImage` file is in the root directory of the main SDK kit.
- `InstallShield 5.53` is installed at `C:\Program Files\InstallShield\InstallShield 5.5 Professional Edition` and `InstallShield PackageForTheWeb` is installed at `C:\ProgramFiles\InstallShield\PackageForTheWeb 2`.
- The environment variables are set as:


```
MK_IS55=c:\progra~1\instal~1\instal~1.5pr
MK_PFTW=c:\progra~1\instal~1\packag~1
```
- The MKS tools are in the path (suppose C:drive is used):


```
C:\>which sh
C:\MKS\MKSNT/sh.exe
C:\>which perl
C:\MKS\MKSNT/perl.exe
C:\>which find
C:\MKS\MKSNT/find.exe
```
- `InstallShield` is installed properly.

CISCO CONFIDENTIAL**Note**

You must make the corresponding changes in these variables, path, and InstallShield installation parameters.

To create the CWCS image, use the sample2.sh script in current directory.

- Launch the sample2.sh file using sh and with four arguments (don't use \ in args, use /):

```
>sh ./sample2.sh arg1 arg2 arg3 arg4
```

where:

- *arg1* is the target directory; let it be for example D:/image directory
- *arg2* is the directory with files myapp.cd.tar and myapp.runtime.tar (actually current directory)
- *arg3* is the directory with buildImage file
- *arg4* is the directory with protopackages and is5.runtime.tar files

This is the sample2.sh file:

```
#####
# INPUT:
# 1 - Target_dir
# 2 - myapp_dir
# 3 - buildImage_dir
# 4 - proto_dir protopackages&is5 directory
#####

if [# -ne 4]; then
    echo "ERROR:sample2 called with insufficient args."
else
    Target_dir=$1
    if [-d $Target_dir]; then
        echo "Directory $Target_dir already exists, remove it first."
    else
        myapp_dir=$2
        buildImage_dir=$3
        proto_dir=$4

        mkdir $Target_dir

        perl $buildImage_dir/buildImage -r -d $Target_dir \
            $proto_dir/is5.runtime.tar $myapp_dir/myapp.runtime.tar \
            $myapp_dir/myapp.cd.tar $proto_dir/cam.runtime.tar \
            $proto_dir/cmf.runtime.tar $proto_dir/cmfw2.runtime.tar \
            $proto_dir/cmfwd.runtime.tar $proto_dir/db.runtime.tar \
            $proto_dir/dmgt.runtime.tar $proto_dir/eds.runtime.tar \
            $proto_dir/ess.runtime.tar $proto_dir/grid.runtime.tar \
            $proto_dir/jawt.runtime.tar $proto_dir/jchart.runtime.tar \
            $proto_dir/jext.runtime.tar $proto_dir/jgl.runtime.tar \
            $proto_dir/jpwr.runtime.tar $proto_dir/jre2.runtime.tar \
            $proto_dir/jrm.runtime.tar $proto_dir/lotusxsl.runtime.tar \
            $proto_dir/nmcs.runtime.tar $proto_dir/perl.runtime.tar \
            $proto_dir/plugin.runtime.tar $proto_dir/pxhlp.runtime.tar \
            $proto_dir/snmp.runtime.tar $proto_dir/svc.runtime.tar \
            $proto_dir/swng.runtime.tar $proto_dir/vorb.runtime.tar \
            $proto_dir/web.runtime.tar $proto_dir/xml4j.runtime.tar \
            $proto_dir/xrts.runtime.tar $proto_dir/eds.cab.tar \
            $proto_dir/jgl.cab.tar $proto_dir/swng2.cab.tar \
            $proto_dir/vorb.cab.tar \

    fi
fi
```

CISCO CONFIDENTIAL

The output of buildImage command is very lengthy. The actual set of tar files may be different from this example. Here is the result:

```
D:\image>ls
<Name of Self-extracting installation file>.exe disk1 temp
autoinstall.sh extract
```

- Now the necessary product (CiscoWorks Home Page, and the customer's application' see the options in ADVANCED_CHOICE section above) can be installed either by running the *Name of Self-extracting installation file.exe* or by running the setup.exe from the d:\image\disk1 directory.



Note To display messages during installation and uninstallation, the myapp.rul file may be prepared and located into myapp\install directory before making the runtime protopackage tar-file.

```
D:\myapp\install>cat myapp.rul

declare
function mypp_preinstall()
begin
MessageBoxLog("Installing MyAP into \n"+NMSROOT, INFORMATION);
end;
function myapp_postinstall()
begin
MessageBoxLog("Running postinstall script for MyAPP",
INFORMATION);
end;
function myapp_uninstall()
begin
MessageBoxLog("Running uninstall script for MyAPP",
INFORMATION);
end;
```

Customizing the Installation Workflow for Windows

The workflow implemented by the CWCS installation framework can be customized for a specific product. You can also modify the workflow to display additional product-specific dialogs at installation time.



Note If you have questions about installation framework customization, contact the CWCS installation team (cmf-install@cisco.com) during the early stages of your project.

The following topics describe how to customize the Windows installation workflow:

- [About the Installer Workflow](#)
- [Getting Started with Windows Installer Tools](#)
- [Creating the Installation Project File](#)
- [Creating Install Actions](#)
- [Creating Install Panels](#)
- [Specifying Conditions For Install Actions and Panels](#)
- [Creating the Install Staging Area](#)

CISCO CONFIDENTIAL

- [Example: Adding Message Boxes to an Installation](#)
- [Example: Creating Custom Password Dialogs](#)
- [Example: Adding User Data to Show Details](#)

For information about limited customizations available on Solaris, see the “[Customizing the Installation Workflow on Solaris](#)” section on page 21-104.

About the Installer Workflow

The installation framework supplies the installer that:

- Loads the table of contents (disk.toc) and package property files (*.info) from the CD image.
- Performs all steps necessary to interact with the end user during the installation process
- Installs all packages, invoking the package-specific xxx_preinstall and xxx_postinstall functions at the appropriate times.

Some products require that additional dialogs be displayed at installation time. If they are implemented in the xxx_preinstall functions, then the dialogs are displayed too late, and might be hidden in the background. This topic describes how dialogs (or non-interactive actions) can be added to the installation of a specific product.

The installer is implemented as a sequence of steps. Each step is either an action or a panel. The sequence is controlled by the project file. When creating an image, the installation framework build tools load the project file and automatically generate the code that invokes actions and panels and controls the sequence, including the processing of the Next / Back buttons required by the wizard model.

Customizing the installation workflow requires the following steps:

1. Write InstallShield functions implementing the step required for the product installation.
2. Add this step to the project file (see the “[Creating the Installation Project File](#)” section on page 21-77).

Getting Started with Windows Installer Tools

To develop InstallShield code that works with the installation framework, be sure your desktop has all required tools. You must have:

- InstallShield Professional 5.5.3 (version 5, maintenance pack 3);
- MKS. In particular, perl, find, and sh utilities must be MKS. You might want to verify the path before trying to build the CD image.

Creating the Installation Project File

The project file is an ASCII file that contains definitions of steps, one step per line. Normally one project file (the template) will contain steps only, with no operations. Other project files will contain additional steps or changes to the template, and use operations with references instructing the generator how to modify the template.

Each step definition uses the following format:

```
[%operation reference][label:]stepName;[sourceFile];condition[;condition[...]]
```

Where:

CISCO CONFIDENTIAL

- `operation` is identified by the `%` sign. The operation can be any one of the following: `AFTER`, `BEFORE`, `REPLACE`, or `DELETE`.
- `reference` specifies where to add the new step or what step needs to be replaced or deleted. The rest of the line describes the step to be added, replaced, or deleted included.
- If `label` is present, it is separated from other parameters by a colon (:). All other fields are separated by semicolons (;).
- Each step must have a `stepName`. If the step name begins with `panel_`, then panel-specific code will be generated (see the “[Creating Install Panels](#)” section on page 21-79). Otherwise, the step is considered an action (see the “[Creating Install Actions](#)” section on page 21-78).
- Following the step is the `sourceFile` (or header) name. The `sourceFile` name must be a file name, not a path name, and can be either the `.h` or `.rul` file.
- The `condition` can be either inline or a function call (see the “[Specifying Conditions For Install Actions and Panels](#)” section on page 21-82). You can have any number of conditions, but you must have at least one.
- Comment lines are allowed, and must begin with `#`.

Creating Install Actions

Installer steps are implemented as InstallShield functions. Functions can be either panels or actions. All panels and actions must have a predefined set of parameters, and return codes for the installer to determine the next step. Each panel or action can be accompanied by one or more conditions. The installer evaluates specified conditions and calls the action or panel only if all conditions are positive.

The source code for actions must be included in one of the protopackages in the `install/action` directory. In the project file, the name of an action is the same as the function, and must be followed by the source file name (without a path). The source file name can be either a header (`.h`), or a source (`.rul`) file.

- If a header file is specified, then the installation framework build tools automatically generate the appropriate include statements for the header file, and a matching include for the source file (replacing `.h` suffix by `.rul` suffix).
- If a source file is specified, the prototype for the function is generated instead of including a header file.

This gives you two options, as follows:

Option 1

Put the source code for the custom action into the `.rul` file with no prototype, and specify the `.rul` file name in the project. For example:

1. Put the source of `myAction` into the `install\action\myActionSrc.rul` file:

```
function myAction()
begin
    ...
end;
```

2. In the project file, add the following:

```
... myAction;myActionSrc.rul
```


CISCO CONFIDENTIAL

Option 2

Put the source code of the custom function into the .rul file and create a matching header file with function prototypes. For example:

1. Put the source of a myAction into install\action\myActionSrc.rul file:

```
function myAction()
begin
    ...
end;
```

2. Put the prototype of a myAction into install\action\myActionSrc.h file:

```
prototype myAction();
```

3. In the project file, add the following:

```
... myAction;myActionSrc.h
```

Normally, Option 1 is more convenient for simple cases, where the action is the only function in the source file. If you want to include multiple functions in a single source file, then Option 2 is the better choice.

Non-Interactive Actions

Non-interactive actions are simple steps that are normally non-interactive, or display simple dialog boxes without Next/Back buttons. Functions implementing such steps must have no parameters, and return codes as follows:

- Return 0 or a positive number to indicate that the step executed successfully and that the installer should proceed to the next step.
- Return a negative number (in the range between -1 and -1000) to indicate that installation must be aborted.

To distinguish actions from panels, the name of non-interactive actions must *not* begin with `panel_`.

Creating Install Panels

Panels are the steps that implement dialogs displayed to the user running the installation. These dialogs use the Next and Back buttons following wizard paradigm. Functions implementing a panel must have:

- A name beginning with `panel_`.
- One numeric parameter (NUMBER). When a panel function is invoked the value of this parameter is the return code of the previously-displayed panel. Usually, it will be either NEXT or BACK depending on what button the user clicked in the previous dialog.



Note This parameter must be checked if the custom panel displays more than one dialog. If the value parameter is BACK, the step must open the last dialog rather than the first.

The return code of a panel must be one of the following:

- NEXT if user clicked on the Next button and installer should proceed to the next dialog.
- BACK if user clicked on the Back button and installer must return to the step displaying previous dialog.
- SM_SKIP if the panel was skipped. This is important to notify the framework that if user decides to go back from the next dialog this dialog need not to be displayed.
- A negative number in the range between -1 and -1000 to indicate that installation must be aborted.

CISCO CONFIDENTIAL

Note The constants NEXT and BACK are defined by InstallShield. The SM_SKIP constant is defined by installation framework.

The implementation of a panel must be able to handle several special cases:

- In silent mode, the installation must proceed without user interaction. If there is a clear default value for the data requested from the user, that value must be set as if the user entered it. In some cases there are no defaults, and information must be loaded from the answer file (see the [“Creating the Answer File”](#) section on page 21-89). The installation framework preloads the name=value pairs from the answer file into the global list lAnswerFile and can be fetched using the getStringFromList function, as shown in [Example 21-3](#).

Example 21-3 Fetching Answers Using getStringFromList

```
if (bQuiet = TRUE) then
    // this is silent mode
    if (getStringPropertyFromList(lAnswerFile,
        "myParameter", svMyParameter) = 0) then
        // myParameter was specified in answerFile, proceed with this value
        ... // some code that puts svMyParameter into temp location for
            // postinstall, see more on that later
    else
        // value is not specified in the answer file. Your postinstall must
        // be able to handle that situation, if it cannot than the default
        // value must be stored here
        // ...
        // In the case when there can be no default, it is time to abort:
        WriteLogFile("ERROR: the value myParameter is not specified in" +
            " the answer file, and silent installation can not proceed");
        return -1; // this instructs installer to abort
    endif;
```

- All dialogs are displayed to the user at the beginning of the installation, and there is always a possibility that the user will decide to move back to an earlier panel or even cancel the installation. Therefore, panel functions must not change anything on the target system. Instead, the data entered by the user should be stored temporarily such that postinstall code can pick up that data and apply it as appropriate. For example:

- Data can be saved in a temporary file under PROP_DIR. This global variable is populated by installation framework with the pathname of a directory under InstallShield's temporary directory. InstallShield will clean up these files once installation finishes. With the addProperty/getProperty functions, you can use:

```
addProperty(PROP_DIR, "myTempFileName", "myParameter", szValue)
```

in the code of the panel, followed by:

```
getProperty(PROP_DIR, "myTempFileName", "myParameter", svValue)
```

in postinstall.

- There are two functions that allow you to pass data from panels to postinstall. You can save data using:

```
ListAddString(lTemp, svData1);
ListAddString(lTemp, svData2);
```

CISCO CONFIDENTIAL

```
SetPassEx(lTemp, "myPanelData");
```

In postinstall, you can retrieve data with:

```
if (GetPassEx("myPanelData", svData1, 0) = 0) then
    // use svData1
else
    // that piece of data was not entered by user either because of silent mode
    // or the dialog was skipped
endif;
if (GetPassEx("myPanelData", svData2, 1) = 0) then
    // the same for the second field
    // ...
```

In addition to passing data, these two functions ensure that temporary files are stored in a directory ciphered for the current user, so no other user can view the information. These functions are therefore safe for passwords and other security sensitive data.

- We recommended that dialogs requesting information from the user be displayed with a Custom installation only. In the Typical installation, dialogs should not be displayed. Therefore, panels would normally be used with conditions. The postinstall code trying to apply the data should provide for cases where data was not saved from the dialog, thus allowing you to preserve existing configurations from previous installations or apply the defaults.
- It is important that conditions excluding panels are specified in the project rather than processed internally by the panel function itself. Consider the following implementation of a panel:

```
function panel_myPanel()
begin
    if (...) then
        // this will cause problem, don't do it!!!
        return NEXT;
    endif;
    // displaying actual panel
    return NEXT;
end;
```

In this case, the framework is unaware that the panel was not actually displayed. After calling the `panel_myPanel` function, the framework would proceed to the next dialog. If the user then clicks on the Back button, the framework will invoke `panel_myPanel` again, which is wrong. In order to avoid this situation, a condition must be specified as a function or inline condition in the project. The previous code can be corrected as follows:

```
function panel_myPanel()
begin
    if (...) then
        // this return code notifies the framework that myPanel
        // was not displayed. When user goes back from the next
        // dialog, the framework would jump to the panel that
        // had been displayed before myPanel
        return SM_SKIP;
    endif;
    // displaying actual panel
    return NEXT;
end;
```

CISCO CONFIDENTIAL**Specifying Conditions For Install Actions and Panels**

Often, installer steps must be skipped, but the decision to execute or skip a step will be made only at installation time. You can allow for this by specifying conditions for actions or panels. The framework evaluates all conditions before executing an action or panel, and the action or panel function is invoked only if *all* conditions pass. Two types of conditions are supported:

- **Condition functions:** You can add these just like custom panels or actions. The function does not have any parameters, and must return a positive number if satisfied, or a negative number to skip the action or panel. For the framework to check a condition, the name of the condition function must be specified in the project file. For example:

```
myAction;myAction.rul;needAction;additionalCondition
```

In this example, the action `myAction` will only be invoked if both the `needAction()` and `additionalCondition()` functions return a positive number.

- **Inline conditions:** These allow simple checks without creating a function. Build tools simply embed such conditions inside an `if` statement. For example, to run a panel in custom mode, only the panel can be registered in the project file:

```
myAction;myAction.rul;svSetupType="custom"
```

Creating the Install Staging Area**Note**

The installation staging area should be available on a local drive. The following examples assume that the staging area is created in the `d:\image` directory. If you want the staging area to be in another directory, adjust the commands accordingly.

There are two ways to create the staging area:

- Copy it from the result area of the NMTG automated builds. You will see the `image\extract` directory under every build. Just copy this directory into `d:\image`.
- Run the `buildImage` command. Copy the command from the log file of an automated build and run it locally, changing the destination directory (the value of the `-d` parameter) to `d:\image`. This command both creates the staging area and builds the installable image in `d:\image\disk1`.

After creating the staging area, you can change the install code and rebuild the image as needed using one of the following options:

- **Option 1:**

```
cd d:\image\extract\is5\install
set debug=1
sh runme.sh 2>&1 | tee d:\image\log.txt
```

These commands rebuild the CD image from the content of `d:/image/extract` area. The process takes about 10 to 20 minutes. Because the output of these commands is very lengthy, it is redirected into the file `log.txt` so it can be analyzed after `runme.sh` finishes. Setting the debug variable causes `runme.sh` to skip creating the self-extracting executable (this will save you 5 to 10 minutes), and adds debug information to the image.

- **Option 2:**

```
set IMAGE=d:\image
d:/image/extract/is5/install/rul.cmd main projectFiles
```

CISCO CONFIDENTIAL

These commands simply rebuild the source code of the installer from the specified project files. The `run.cmd` file can be copied into the directory in the path to avoid typing the path each and every time.

The `projectFiles` variable requires that the path names of one or more project files be relative to the `d:\image\extract` directory. For example, to build the installer based on the CWCS project with additional steps in the file `d:\image\extract\kilner\install\action\kilner.project`, use the following command:

```
run main cmf\install\action\core.project
      kilner\install\action\kilner.project
```

This method is available only after `buildImage` or `runme.sh` builds a complete image .

Example: Adding Message Boxes to an Installation

To add a simple message box to a CWCS installation:

Step 1 Copy the extract area from the CWCS daily build into the `d:\image` directory.

Step 2 Build an image in your staging area. To do that, run the following commands:

```
cd d:\image\extract\is5\install
set debug=1
sh runme.sh 2>&1 | tee d:\image\log.txt
```

When `runme.sh` finishes, the installable image is created at `d:\image\disk1`. You can run the beginning of installation and cancel it when the Installation Type dialog is displayed. Please note the sequence of dialog boxes before it.

Step 3 Create the function displaying the custom message box. For example, using any text editor, create the file `d:\image\extract\cmf\install\action\helloWorld.rul`, as follows:

```
// helloWorld.rul - my first custom action
function helloWorld()
begin
    MessageBoxLog("Hello World", INFORMATION);
    return 0;
end;
```



Note In this example, additional files are created in the CWCS package area. For actual projects, additional files with custom actions and projects should be added to the packages of the product *using* CWCS rather than CWCS itself.



Note You must create actions and projects under the `extract\<pkg>\install\action` directories. This way, `buildImage` or `runme.sh` can locate such files when the image is built.

Step 4 Add the action to the project. For example, using any text editor, create the file `d:\image\extract\cmf\install\action\my.project` and add the Hello World `rul` file to it, as follows:

```
project adding HelloWorld to CWCS installation
%AFTER panel_welcome helloWorld;helloWorld.rul
```

Step 5 Rebuild the installer. Run the following commands:

```
set IMAGE=d:\image
run main cmf\install\action\core.project cmf\install\action\my.project
```

CISCO CONFIDENTIAL

- Step 6** Now run the installation again. You should see the dialog box with the Hello World message immediately following the Welcome dialog.

Example: Creating Custom Password Dialogs

This example describes how to add a dialog that prompts the user to type a password. It is assumed that CWCS is included in the CD with the product image, and therefore all CiscoWorks dialogs need to be displayed. The dialog described here will be added in such a way that it is displayed by a custom installation only. If the product is installed for the first time, a random password must be generated.

The panel will be implemented by the `panel_myPwd` function. The file with this function will be added to the `install\action\mypassword.rul`, and the prototype will be added to the `install\action\mypassword.h`.



Note This example assumes that the installable image of the product is already being built and includes the product protopackages as well as the CWCS protopackages. The `image/extract` directory must be copied to `d:\image` directory.

- Step 1** Create the installable CD:

```
cd d:\image\extract\is5\install
set debug=1
sh runme.sh 2>&1 | tee d:\image\log.txt
```

- Step 2** Add the files `mypassword.h` and `mypassword.rul` to one of protopackages (for example, `myPkg`). There is no need to create new protopackages; just use one of existing product protopackages.

The file `extract\myPkg\install\action\mypassword.h` contains one line:

```
prototype panel_myPwd(NUMBER);
```

To add the panel to the installer, a new project file is needed (for example, `extract\myPkg\install\action\my.project`). This project file needs the following line (do not put extra spaces at the start):

```
%BEFORE addUserInputToInfoList
    panel_myPwd;mypassword.h;svSetupType="custom"
```

This condition ensures that the panel is displayed in custom mode only.

The rest of the code goes into the `extract\myPkg\install\action\mypassword.rul` file. It contains the code for the `panel_myPkg` function:

```
function panel_myPwd(lastStepRc)
    NUMBER nRc;
    STRING szPwd;
    LIST  lLabels, lTemp;

    begin
        if (bQuiet) then
            // this is quiet mode.
            if (IsInstalled("myPkg") = 0) then
                // this is reinstallation case, leave it to postinstall to handle
                WriteLogFile("INFO: Reinstallation of myPkg, skipping password");
                return NEXT;
            else
                // original installation, let's try answer file
```

CISCO CONFIDENTIAL

```

if (getStringPropertyFromList(lAnswerFile,
    "myPassword", szPwd) != 0) then
    // generate random password
    randomWord(szPwd, 15, -2, RW_ALL);
endif;
// save the password for postinstall to pick up
lTemp = ListCreate(STRINGLIST);
ListAddString(lTemp, szPwd, AFTER);
SetPassEx(lTemp, "myPwd");
ListDestroy(lTemp);
return NEXT;
endif;
endif;

// let's do the dialog
// here you might want to reinitialize the szPwd with the value of
// existing password in the case of reinstallation / upgrade

lLabels = ListCreate(STRINGLIST);
ListAddString(lLabels, "My Password", AFTER);
nRc = skPassExTitle(
    "Description",
    "text of comment", PASS_DIALOG_ID,"PASS_DIALOG_ID","", "myPwd",
    PASS_EX_DB_VALIDATION + PASS_EX_EMPTY_ALLOWED + 15,
    lLabels, 0, "Change My Password");
ListDestroy(lLabels);
return nRc;
end;

```

Example: Adding User Data to Show Details

In the CWCS installation, information collected from the user or generated automatically is displayed in the Confirmation Dialog, which allows the user to review the options. This dialog is implemented by `panel_installConfirmation`. This step generates the standard portion (the one displayed while the details are hidden) and appends the details from the global list `IUserOptions`. This list is managed by the installation framework; the action `addUserInputToInfoList` populates CWCS-specific options.

To insert application-specific options into the details, you can either replace `addUserInputToInfoList` or add another step that appends the application-specific options to the list.

For example, to add the password created by `panel_myPwd`, create this action:

```

/*-----
 * adds the password to details
 *-----*/
function addMyPwdToInfoList()
STRING svTemp[1024];
begin
    if (GetPassEx("myPwd", svTemp, 0) = 0 && svTemp != "") then
        ListSetIndex(lUserOptions, LISTLAST);
        ListAddString(lUserOptions, " My password: " + svTemp, AFTER);
    endif;
end;

```



Note This function must not create the `IUserInfoList`, but should clean it up.

To add this step to the installation, put the following line into your project file (where the function is in `mypassword.rul`, and prototype is in `mypassword.h`):

CISCO CONFIDENTIAL

```
%AFTER addUserInputToInfoList addMyPwdToInfoList;mypassword.h
```

To use application-specific information instead of that created by CWCS, or when your installation does not include CWCS, the code must be modified slightly. For example, to add the password created by panel_myPwd to a non-CWCS installation, or when replacing the details provided by CWCS, create an action that looks like this:

```
/*-----
 * adds the password to details
 *-----*/
function addMyPwdToInfoList()
STRING svTemp[1024];
begin
    EmptyList(lUserOptions);
    if (GetPassEx("myPwd", svTemp, 0) = 0 && svTemp != "") then
        ListSetIndex(lUserOptions, LISTLAST);
        ListAddString(lUserOptions, " My password: " + svTemp, AFTER);
    endif;
end;
```



Note This function must not create the lUserInfoList or clean up the list. These tasks are performed by other steps in the installation.

To add this step to a non-CWCS installation, add the following line to your project file (where the function is in mypassword.rul, and prototype is in mypassword.h):

```
%BEFORE panel_installConfirmation addMyPwdToInfoList;mypassword.h
```

To add this step to CWCS installation in which the CWCS details are overridden, add the following line to your project file (where the function is in mypassword.rul, and prototype is in mypassword.h):

```
%REPLACE addUserInputToInfoList addMyPwdToInfoList;mypassword.h
```

Solaris Installation Reference

This topic covers the following reference information for Solaris:

- [Setting Ownership for Package Files on Solaris](#)
- [Creating the Answer File](#)
- [Writing Solaris Scripts](#)
- [Using the Solaris Installation APIs](#)
- [Using Solaris Build Tools](#)
- [Solaris Getting Started Example](#)

Setting Ownership for Package Files on Solaris

The CWCS team has updated installation on Solaris to set the ownership of application files properly during CWCS installation. If you have application files that existed in a previous release and were not part of the installation package, then you must modify the post-install hooks/scripts to add chown/chgrp commands for each file. This includes dynamically created files such as data files and properties files.

The options for setting ownership include:

CISCO CONFIDENTIAL

- Fixed user ownership for all packages in the build
- Fixed user ownership for one package

This section provides information about setting ownership data during the image build process using the `buildImage` script. The `buildImage` script expands the `protpackages` and searches for the `runme.sh` file and runs it. On Solaris, the `.../install/runme.sh` comes from the `pkgtools` `protpackage` and performs the following tasks:

- It assembles the installable image from expanded `protpackages`.
- It calls the `makesolpkg` script to build Solaris packages. During `makesolpkg` execution for each package, the package content description file called “prototype” is created with file ownership setting as the result of the following procedure.

At the beginning of its execution, the script `makesolpkg` has the `bin:bin` ownership hard coded for `$OWNER` and `$GROUP` variables.

Additional facilities have been developed to supply you with the ability to have files owned, as they must be set in the image. These are getting ownership from `package_name.owner` file (for every package). The ownership setting sequence for particular package:

- If the `package_name.owner` file exists, then the ownership is assigned in accordance with the data from this file.
- If there is no `package_name.owner` file, then ownership is assigned as `bin:bin` from `$OWNER` and `$GROUP` variables.

The prototype file is used by `pkgmk` utility for package creation.

Related Topics

See the:

- [“Setting Ownership from package_name.owner File” section on page 21-87.](#)
- [“Setting Ownership During Build/Installation” section on page 21-88.](#)
- [“Setting Ownership Assignment Details” section on page 21-88.](#)
- [“Understanding and Implementing the casuser” section on page 21-21.](#)

Setting Ownership from `package_name.owner` File

The `package_name.owner` file (see the [“Understanding the package_name.owner File” section on page 21-88](#)) is a manifest-like file where you can specify the ownership for files/directories with wild cards. It is necessary to have a separate ownership description file for each package. The rules for this file are:

1. The first string of this file is the header: `# This file was automatically created with tocToOwner.pl.`
2. The second string contains: `DEFAULT_OWNER=name`
`DEFAULT_GROUP=default_owner's_group`

These two items specify the default owner/group for all the files in the package. This data may be set for any file/directory in the package by explicitly mentioning it in BOM file or by using `PROP_IMAGE_OWNER` and `PROP_IMAGE_GROUP` build environment variables.

3. The remaining strings have the structure: `directory/file_name owner_login owner_group`

These items should be separated by spaces. You can use wild cards for directories (`*-s` at the end of `directory_name` string; no any space symbols are allowed between the end of directory name and `'*-symbol`). One symbol (`*`) refers to the current directory only; the ownership described by

CISCO CONFIDENTIAL

owner_login/owner_group is to be assigned for every file/directory in current directory only. Two symbol ** string means recursion; the ownership provided by *owner_login/owner_group* is to be assigned for every descendant file/directory starting from those in the current directory.

Understanding the *package_name.owner* File

The protopackages are built as specified by the BOM files. Each package's BOM file provides the ownership data. The most convenient way to set up your .owner file is to use the appropriate .toc file. The .toc file is automatically produced from .bom file during the build procedure and contains more detailed data about your directories/files ownership.

An appropriate script (tocToOwner.pl) was written and the .owner file is now automatically put into protopackage tar-file (along with .bom file and others). So the *package_name.owner* file is automatically located into right `./install` directory.

Setting Ownership During Build/Installation

This topic provides information on typical scenarios for you to set desired ownership during image creating process or during installation. The typical scenarios are:

- Fixed user ownership for all the packages in the build

To implement this scenario, supply the input parameters OWNER/GROUP in the *pkg_name.owner* file. The makesolpkg file can then get them from DEFAULT_OWNER/DEFAULT_GROUP tags in *pkg_name.owner* file.

This scenario may be used to set casuser:casuser ownership for all the packages.
- Fixed user ownership for one package

There are two possibilities for this scenario. First, specify fixed ownership in the .bom file for particular package. Second, prepare the .owner file manually and deliver it to appropriate `/install` directory during image build.

Setting Ownership Assignment Details

This procedure is implemented by the makesolpkg script. The ownership assignment rule works in the following manner.

- If any particular substring *directory/file name* of string A in *package_name.owner* file matches after applying wildcard rules described above to some directory/file mentioned in the "prototype" file for some package, then the ownership from this string A is assigned to the directory/file in prototype file.
- When there are few possible matches, only the last one from *package_name.owner* file is assigned.
- When no matches are valid for some directory/file in the package, the data from DEFAULT_OWNER/DEFAULT_GROUP items of *package_name.owner* file is used for this directory/file.
- When an absent DEFAULT_OWNER/DEFAULT_GROUP string in *package_name.owner* file (or the absence of this file), the hard coded data is used for ownership assignment (currently it is bin:bin).

The makesolpkg script is located in pkgtools.runtime.tar file. It is untared into the `./pkgtools/install` directory. The package is created in two steps:

1. First, the ownership description file prototype is created with pkgproto utility

CISCO CONFIDENTIAL

2. The ownership description file is used as input for pkgmk utility for package build.

After the execution of pkgproto, the prototype file is modified so that the ownership is changed in accordance with ownership assignment rule. Then it creates the package file with the new ownership.

Creating the Answer File

The answer file is an ASCII file that provides the required inputs for quiet installations. It contains the name=value pairs shown in [Table 21-20](#).

**Caution**

For CWCS, the answer file is required because it contains the mandatory `adminPassword` field. If this file is not present for CWCS, setup will exit.

The answer file contains the following name=value pairs:

Table 21-20 Answer File Properties

Property	Description
destination	Optional. Allows quiet installation to install into a directory other than <code>/opt/CSCOpX</code> . If not specified, installation goes into <code>/opt/CSCOpX</code> .
adminPassword	Required for CWCS. For other products, if there are no mandatory fields, the answer file is not required. Specifies the login password for the admin user. It is only used for original installations; reinstallation in quiet mode leaves the password alone.
secretPassword	Specifies the login password for the secret user. It is only used for original installations; reinstallation in quiet mode leaves the password alone.

Notes

- To install a product in quiet mode *when an answer file is not required*, use this command:

```
./setup.sh -q ""
```

In this example, the blank quotes are required.

Example

Answer file:

```
#cat /tmp/answer_file
##Sample Answer file
destination=/opt/cisco
adminPassword=adminpass
```

Setup command:

```
#setup.sh -q /tmp/cscotmp/enpass_cmf
```

Related Topics

For the same procedure on Windows platforms, see the [“Preloading the Global List, IAnswerFile” section on page 21-35](#).

CISCO CONFIDENTIAL

Writing Solaris Scripts

Using the CWCS installation APIs, you can write Bourne shell scripts that will allow you to specify any requirements and enforce any constraints on a package on a Solaris platform.

The following topics describe how to write scripts for your package:

- [Using the Solaris Installation Hooks](#)
- [Where to Find Solaris Installation Examples](#)

Using the Solaris Installation Hooks

Solaris developers can use the following Bourne shell scripts to perform the required functions for their package installation:

Table 21-21 *Solaris Hooks*

Hook Type	Description
preinstall	This hook is run by installer before file transfer. Most packages do not need this script. If you need your software to do something (for example, verify prerequisite software) before installing your files, then you need this script.
postinstall	This hook is run after file transfer. It contains the actual configuration of the component. This is the correct place to register daemons.
preremove	This hook is run when the component is uninstalled immediately before removing files. This hook is a good place to unregister daemons or verify that the package is installed.
postremove	This hook is run when the component is uninstalled immediately after removing files. Use this hook to clean up files and directories that are not registered with the packaging system. For example, use this script to remove log directories and files.
prerequisite	This hook is run before any preinstall is performed. It specifies dependencies on other packages, disk space required, and questions to be asked of the installer. If the information obtained by this script is required by another script, such as preinstall, then use AddProperty and GetProperty APIs. If the prerequisite hook is set for the installation to fail, it should return a non-zero return code.



Note Do not ask the user any questions in these files: preinstall; postinstall; preremove and postremove. Instead, define shell variables with reasonable defaults. The installation script will call the package's prerequisite script to ask the questions and set the variables for the other scripts.

If the information obtained by this script is required by another script, such as preinstall, then the script may set an environment variable or write a temporary file. The preferred method is to call AddProperty (see the “AddProperty” section on page 21-98) to save the value and GetProperty (see the “GetProperty” section on page 21-98) in a later script to retrieve the property. These functions are found in commonscript.sh.

Hooks are run in the following sequence:

1. Each prerequisite for all packages.

CISCO CONFIDENTIAL

2. pkgadd for all components. Each pkgadd executes preinstall, file transfer, and postinstall for this package.

At uninstallation time, hooks execute pkgm for each component. Each pkgm executes preremove, removes files, postremove.

It is important to understand in which order hooks are executed. The installer observes the following rules at installation:

- The installer skips installation of a component if it decides that a particular component should not be processed. It leaves out a component if its pending version is lower than the installed version. It can leave out the component if it is optional and not required by any other component.
- The installer does not execute a skipped component's prerequisite hook. All other prerequisite hooks are executed before any preinstall is executed.

At uninstallation the installer observes the following rules:

- Shared packages are not removed and their uninstall hooks are not executed unless all installable units they belong to are uninstalled.



Note

In general, hooks should not rely on any order of execution. This is especially important in order to provide the same hooks for both major releases as well as for upgrades.

Additional hooks are provided to permit custom operations at the completion of various install operations. For details, see the [“Customizing the Installation Workflow on Solaris”](#) section on page 21-104.

Where to Find Solaris Installation Examples

The following installation examples are documented:

- Prerequisite
- Preinstall
- Postinstall

A basic example is provided in the [“Solaris Getting Started Example”](#) section on page 21-106.

Using the Solaris Installation APIs

This topic covers the Solaris installation APIs, which you can use in in any hook:

- [Using the Solaris Input/Output APIs](#)
- [Using the Solaris Package APIs](#)
- [Using the Solaris System APIs](#)
- [Using the Solaris Installable Unit APIs](#)



Note

To use Solaris APIs, add the following line at the beginning of hooks. Remember to use the period at the beginning of the line: `. ${SETUPDIR}/commonscript.sh`

The following is a sample function header:

```
#####
```

CISCO CONFIDENTIAL

```
## FUNCTION FunctionName <input> [<input>]
## brief description of function.
#####
```

The FUNCTION line shows the function name and all arguments with which it is called.

**Note**

Unless otherwise noted, all functions return 0 for success and 1 for failure.

Using the Solaris Input/Output APIs

The Input/Output APIs handle password management and input/output functions:

- **AskPass**: Prompts for your password
- **AskPassEx**: Prompts for your password
- **GetPass**: Retrieves the password from a temporary file
- **ChangeDbPasswd.pl**: Changes the Database password in the database and properties file after the installation is completed
- **AddPassword**: Used at the end of the installation to display the passwords that were either entered or randomly generated.
- **PasswordRandomSelection**: Returns a randomly-generated password.
- **PortRandomSelection**: Returns a randomly-generated port number.
- **Profile**: Returns a randomly-generated port number.
- **Debug**: Returns a randomly-generated port number.
- **PromptResponse**: Returns a randomly-generated port number.
- **PromptYN**: Returns a randomly-generated port number.

AskPass

```
AskPass PromptString Temp_File_Name DbPass PassName
```

Normally, this API is called in the prerequisite to the installation process. The AskPass API:

1. Prompts you for admin and guest passwords during installation.
This API asks the password twice to confirm the password.
2. Validates the password, encrypts it and stores the password in a temporary file in the temp directory.

**Note**

This is a shell script function that is available in commonscript.sh.

Arguments

Field	Description
PromptString	Prompt which asks the user for password.
Temp_File_Name	Name of the file where the encrypted password will be stored. This file will be created in the temp directory (/tmp/cscotmp/<temp_filename> directory).

CISCO CONFIDENTIAL

Field	Description
DbPass	<p>Indicates whether to validate the password.</p> <p>When set to 1: The password is validated according to database rules such as, The password should not start with a number, It should not contain any special characters and password length should not be more than 15 characters. These constraints are imposed by the Sybase DBD.</p> <p>Note If the password is for a database, set to 1.</p> <p>When set to values other than 1: The password is not validated.</p>
PassName	The name of the password. This name is displayed at the end of the installation process using the AddPassword API (see the “AddPassword” section on page 21-96).

Return Values

None

Example

```
AskPass "Enter the CWCS Database Password" "enpass_cmf" 1
        "VALUE_CCSDB_PASSWORD"
```

In this example, the API:

1. Displays the string “Enter the Password” and reads the password. The typed characters will not be displayed on the screen.
2. Prompts the user to retype the password to confirm.
3. If both the password matches and passes the validation, the password will be encrypted and stored in /var/tmp/enpass_cmf file. If the passwords do not match the user will be prompted again for the password. If the password does not pass the validation, an appropriate message is displayed and the user will be prompted again for the password.

AskPassEx

```
AskPassEx prompt_string temp_filename pass_Name flag
```

This API will be normally called in the prerequisite to the installation process. The AskPassEx API:

1. Prompts users for passwords .
2. Prompts for the password twice, to confirm it.
3. Accepts special characters, such as punctuation marks, ampersands, and so on.
4. Validates the password, encrypts it, and stores it in a temporary file in the temp directory.

**Note**

This is a shell script function that is available in commonscript.sh.

CISCO CONFIDENTIAL**Arguments**

Argument	Description
<i>prompt_string</i>	Prompt which asks the user for a password.
<i>temp_filename</i>	Name of the file where the encrypted password will be stored. This file will be created in the temp directory (/tmp/cscotmp/).
<i>pass_Name</i>	Name of the password.
<i>flag</i>	Indicates whether to bypass password validation if the user responds with no entry (1) or reprompt for the password to confirm (0; this is the default).

Return Values

None.

Example

```
AskPassEx "Enter the CiscoWorks admin password: " "enpass_admin" "Admin Password" 0
```

In this example, the API:

1. Displays the user prompt string `Enter the CiscoWorks admin password:.`
2. Reads the password the user enters, and stores it in the file `enpass_admin`. The typed characters are not displayed on the screen.
3. Prompts the user to retype the password to confirm.
4. Assigns “Admin Password” as the name of the password. This name is displayed at the end of the installation process using the `GetPass` API (see the [“GetPass” section on page 21-94](#)).

GetPass

```
retVal=GetPass Temp_File_Name
```

This API is used to retrieve the password from the temporary file. This API also deletes the password file.

**Note**

This is a shell script function that is available in `commonsript.sh`.

Arguments

Field	Description
Temp_File_Name	Name of the temp file where the encrypted password is kept. The file should be present in /tmp/cscotmp directory.

Return Values

Field	Description
retVal	0 = success. If successful, the global variable <code>\$PASS</code> contains the clear text password. 1 = failure

CISCO CONFIDENTIAL**Example**

```
retVal=`GetPass enpass_cmf`
```

In this example, the API reads the encrypted password from /tmp/cscotmp/enpass_cmf file, decrypts it and returns to the variable Pass.

ChangeDbPasswd.pl

`$DBSWDIR/conf /ChangeDbPasswd.pl DSN Password`

This API changes the password in the database and in property files. It accepts the DSN name and password as the parameters and changes the password. This function returns 0 if it is successful, and returns a value greater than 0 (> 0) if it has failed.

The API does the following:

- Checks whether the DSN is valid.
- If the DSN is valid, checks whether the database is enabled.
- If the database is enabled, changes the password in the database, `odbc.templ`, `.odbc.ini`, `DBServer.properties` and the property file specified in `odbc.templ`.
- If the database is not enabled, changes the password in the database and `odbc.templ`.

This API is normally called in `postinstall`.

**Note**

This is a Perl function that is available in `$NMSROOT/objects/db/conf/` directory.

Arguments

Field	Description
DSN	Database name
Password	New password

Return Values

None

Examples

This example changes the password of the CWCS database to “cisco”.

```
$DBSWDIR/conf /ChangeDbPasswd.pl cwcs cisco
```

The following example shows how to change the database password for the CWCS database in the CSCODB package.

1. PromptYN “Do you want to Change the CiscoWorks Database Password?” ‘n’. This API asks the user whether the password need to be changed.
2. If the user responds with “yes”, continue as follows:

```
if [ ${YN} = 'y' ]; then
    AskPass "Enter the CWCS Database Password" "enpass_cmf" 1
```

```
fi
```

3. In the CSCODB package `postinstall`:

CISCO CONFIDENTIAL

```
retVal=`GetPass enpass_cmf`
$DBSWDIR/conf /ChangeDbPasswd.pl cmf $PASS
```

AddPassword

```
AddPassword "password_name" "password_value"
```

This API is used at the end of the installation to display the passwords that were either entered or randomly generated.



Note This utility was introduced in CWCS 2.2. It is a shell script function that is available in `commonsript.sh`.

Arguments

Field	Description
password_name	Name of the password
password_value	Value assigned to password name (either entered or randomly-generated)

Return Values

None

Example

```
AddPassword "VALUE_CCSDB_PASSWORD" "$PASSWORD"
```

PasswordRandomSelection

```
new_password = "PasswordRandomSelection"
```

This API returns a randomly-generated password. It is used when user interaction is not required to enter the password.



Note This utility was introduced in CWCS 2.2. It is a shell script function that is available in `commonsript.sh`.

Valid Installation Modes

Typical

Arguments

None

Return Values

A randomly-generated string

Example

```
PASSWORD="PasswordRandomSelection"
```

CISCO CONFIDENTIAL**PortRandomSelection**

```
port_number = "PortRandomSelection"
```

This API returns a randomly-generated port number. It is used when the designated port is already in use by another process.



Note This utility was introduced in CWCS 2.2. It is a shell script function that is available in `commonscript.sh`.

Valid Installation Modes

Typical

Arguments

None

Return Values

A randomly-generated port

Example

```
port_no="PortRandomSelection"
```

Profile

```
Profile label [output file]
```

If in debug mode, write the time stamp for profiling. Concatenate to a file if one is specified.

Debug

```
Debug output string [output file]
```

If in debug mode, write the string. Concatenate to a file if one is specified.

PromptResponse

```
PromptResponse prompt string default string
```

Prompts user for input and store response in variable `RESPONSE`.

PromptYN

```
PromptYN prompt string default string
```

Prompts user for a yes or no responses and stores the response in lower case in the variable `YN`.

Using the Solaris Package APIs

This group of APIs contains package functions:

- [AddProperty](#)
- [GetProperty](#)

CISCO CONFIDENTIAL

- UpdateAnsFile
- RunRequestScript
- SetInstallMode_SOL
- UpdateInvFile
- Installf
- CheckPkgInstalled
- GetPkgParam
- needsMoreDiskSpace
- SetInstallPkgMode_SOL
- SetInstPkgMode_frmPrePst

AddProperty

AddProperty package name property name property value

Adds a property to a package.

GetProperty

GetProperty package name property name

Retrieves a package's property and stores its value in variable PROPVALUE.

UpdateAnsFile

UpdateAnsFile package name

Calls this function at the beginning of postinstall.

RunRequestScript

Obsolete.

SetInstallMode_SOL

Obsolete.

UpdateInvFile

UpdateInvFile *fileset path permissions uid gid*

Updates the Fileset.inventory file. This function is necessary if you need to modify the attributes of files or directories that are part of the distribution.

Arguments

Field	Description
fileset	Fileset being processed
path	Installer to be modified. This is used as a key to locate a record.

CISCO CONFIDENTIAL

Field	Description
permissions	Permissions.
uid	Owner.
gid	Group.

Installf

This is a platform-independent wrapper providing the functionality of the Solaris install function (installf). For parameters, refer to the Solaris man page on installf.

Use this inside pre- or post- installs if you are modifying or updating data in your package and must ensure that your changes are valid.

CheckPkgInstalled

`CheckPkgInstalled packagename`
Verifies the existence of a package.

GetPkgParam

`GetPkgParam packagename parametername`

Gets the value of the parameter for the package. This is an abstraction of `pkgparam -v pkg param_name` for Solaris.

needsMoreDiskSpace

`needsMoreDiskSpace packagename`

Gets the numeric value of the dynamically determined disk space required for installation and the estimated size required for use in MB from `pkg.nmds`.

SetInstallPkgMode_SOL

`SetInstallPkgMode_SOL package_name`

This API sets the package installation mode after checking if that the package already exists.



Note Use this utility only in the prerequisite to the installation process.



Note This utility was introduced in CWCS 2.2 It is a shell script function that is available in `commonsript.sh`.

Arguments

Field	Description
package_name	The name of the package

CISCO CONFIDENTIAL**Return Values**

Field	Description
PKG_I_MODE	Installation mode options: <ul style="list-style-type: none"> • NEW • REINSTALL • UPGRADE

Example

```
SetInstallPkgMode_SOL CSCOapch
```

SetInstPkgMode_frmPrePst

```
SetInstPkgMode_frmPrePst packagename
```

This API sets the package installation mode after checking if the specified package already exists.



Note Use this utility only in preinstall and postinstall.



Note This utility was introduced in CWCS 2.2 It is a shell script function that is available in commonscript.sh.

Arguments

Field	Description
<i>packagename</i>	The name of the package

Return Values

Field	Description
PKG_I_MODE	Installation mode options: <ul style="list-style-type: none"> • NEW • REINSTALL • UPGRADE

Example

```
SetInstPkgMode_frmPrePst CSCOapch
```

Using the Solaris System APIs

This group of APIs includes system functions:

- [GetBootScript](#)

CISCO CONFIDENTIAL

- [GetInitDir](#)
- [GetLibPath](#)
- [NetstatForPort](#)
- [PortUsed](#)
- [SaveBackFile](#)
- [RestoreBackFile](#)
- [RemoveBackFile](#)
- [DelServices](#)
- [AddServices](#)
- [GetDF](#)
- [GetFreeDF](#)
- [MakeDir](#)
- [GetOS](#)

GetBootScript

Retrieves a package's boot script.

GetInitDir

Retrieves a package's initial directory.

GetLibPath

Retrieves the library path.

NetstatForPort

`NetstatForPort portnumber`

Verifies that *portnumber* is used. Returns 1 if port is in use, else 0.

PortUsed

`PortUsed portnumber tcp or udp`

Verifies that a port is used. Return 1 if port in use, else 0.

SaveBackFile

`SaveBackFile file name`

Saves a copy of file. Use before editing the file.

RestoreBackFile

`RestoreBackFile file name`

Restores a copy of file. Use when editing has failed.

CISCO CONFIDENTIAL**RemoveBackFile**

```
RemoveBackFile file name
```

Removes a copy of this file.

DelServices

```
DelServices service name
```

Removes an entry from /etc/services.

AddServices

```
AddServices service name service port service protocol [comment]
```

Adds an entry to /etc/services.

GetDF

```
GetDF file system
```

Raw output from df or bdf command.

GetFreeDF

```
GetFreeDF file system
```

Gets the numeric value of the disk space for this file system.



Note GetFreeDF expects you to specify the name of the file system , but does not check that you have done so. If you fail to specify a file system, it will get the numeric value of the disk space for the file system last returned by the df -k command.

MakeDir

```
MakeDir directory name
```

Ensures that the proposed directory exists. If not, creates the directory and sets the owner and group to **casuser**. If the directory exists, MakeDir assumes that the owner and group are **casuser**.

GetOS

Sets the environment variables LC_OS and OS to our canonical form of the OS name.

Using the Solaris Installable Unit APIs

These APIs include the installable unit functions:

- [GetNMSRoot](#)
- [SetIMode](#)
- [GetIMode](#)

CISCO CONFIDENTIAL

GetNMSRoot

Retrieves the previously stored value of NMSROOT.

SetIMode

```
SetIMode install mode
```

Stores the value of NMSROOT.

GetIMode

Stores the value of I_MODE.

Using Solaris Build Tools

This topic provides instructions to build an image from protopackages using the installation framework on Solaris. The main steps are:

- [Step 1: Install Third-Party Tools On Solaris](#)
- [Step 2: Install the Framework On Solaris Platforms](#)
- [Step 3: Prepare the Make Image on Solaris](#)

The following topics provide additional guidelines and examples:

- [Customizing the Installation Workflow on Solaris](#)
- [Debugging on Solaris](#)
- [Verifying Packages on Solaris](#)
- [Solaris Getting Started Example](#)

Step 1: Install Third-Party Tools On Solaris

Install the Solaris-specific third-party tools referenced in the [“Including Files in the Protopackage” section on page 21-20](#).

Step 2: Install the Framework On Solaris Platforms

To install the framework, copy the following files to the Solaris working directory:

- `buildImage`: Main script which creates the CD image from protopackages
- `verifyImage`: Script that verifies the structure of the protopackages.
- `is5.runtime.tar`: Protopackage containing the Windows installation framework.
- `pkgtools.runtime.tar`: Protopackage containing the Solaris installation framework.
- `makesolpkg`: Solaris -only script

Step 3: Prepare the Make Image on Solaris

Follow these steps to make the image on Solaris:

CISCO CONFIDENTIAL

Step 1 Verify that MKS is in your path.

```
which sh
which perl
perl -v
```

The output displayed for these commands should show valid paths for each tool. Perl needs to be version 5.5 or higher.

Step 2 Run buildImage command:

```
perl buildImage -r -d image_path toolpath/pkgtools.runtime.tar rtpath/myPkg.runtime.tar
protopackages
```

Where:

- `-r` is the option needed to refresh the `image_path` (by first removing all directories from `image_path/extract` and then extracting all protopackages).
- `image_path` is the full path name of the directory where the image will be created. Normally, free space of at least three times the size of runtime will be required. You must specify the path starting from the top level directory, with forward slashes, no spaces allowed.
- `toolpath` is the full path to the `pkgtools.runtime.tar` file.
- `rtpath` is the full path to your `runtime.tar` file.
- `protopackages` is a space-delimited list of all protopackages to be installed. Each protopackage specification must include full path names with forward slashes, no spaces allowed.

This command will create the following directories under `image_path`:

- `extract` subdirectory: Contains all files extracted from protopackages.
- `disk1` directory: Contains the installable image.

Customizing the Installation Workflow on Solaris

The CWCS installation framework workflow on Solaris does not offer the extensive custom action and panel features available on Windows (for details on these features, see [“Customizing the Installation Workflow for Windows”](#) section on page 21-76).

However, the Solaris framework does offer two features that allow you to customize most installation workflows with a great deal of flexibility, as explained in the following topics:

- [Collecting User Interactions at the Start of a Solaris Install](#)
- [Executing Custom Operations During a Solaris Install](#)

Collecting User Interactions at the Start of a Solaris Install

You can streamline the user’s workflow and make possible “unattended” Solaris installs by prompting the user for all installation inputs at the start of the install, instead of during each install phase when they are required. To do so, simply add the following tag to the corresponding info file:

```
USER_INTERACTION=TRUE
```

You may want to change the wording of the install prompts to ensure that they make sense to users when presented in a single group.

CISCO CONFIDENTIAL

Executing Custom Operations During a Solaris Install

To execute custom operations at various steps during the install, using the hooks shown in [Table 21-22](#).

Table 21-22 **Solaris Workflow Customization Hooks**

Hook	Executes Your Script
AFTER_LICENSE	After license display
AFTER_INSTALL_TYPE	After the user selects the install type
AFTER_PREREQS	After running prerequisites
AFTER_PKG_ADDS	After package additions

You can specify conditions, titles and scripts for these operations much as you do for action customizations on Windows. For example, you can add the following section to your `dsk.toc` file to execute the script `rmJrunWeb.sh` after performing a prerequisite test to ensure that a version of CWCS earlier than 3.0 is present:

```
[AFTER_PREREQS]
AFTER_PREREQS_0=UNINSTALL_JRE2_JRUN_WEB

[UNINSTALL_JRE2_JRUN_WEB]
AFTER_PREREQS_0_CONDITION=cmfwd.2.1.0-2.2.9
AFTER_PREREQS_0_title=Uninstall CSCOjrun CSCOweb CSCOjext CSCOjre2
AFTER_PREREQS_0_script=rmJrunWeb.sh
```

In this example, the script will execute only after prerequisites are checked, and only if the condition is met. The title will be displayed before the script is executed.

Debugging on Solaris

To debug on Solaris, use generic shells. No additional coding is needed. Use `set -x` to add generic debugging information to your log file.

Verifying Packages on Solaris

The Solaris version of the CWCS installation framework provides the `pkgchk` command to permit package integrity verification. You can use `pkgchk` to perform package verification:

- Before installation, within the image.
- After installation.

To verify a package before installation, run `pkgchk` as follows:

```
pkgchk -d location name
```

Where:

- *location* is the package location within the image.
- *name* is the package name.

For example :

```
pkgchk -d disk1/packages CSCOapch
```

CISCO CONFIDENTIAL

If a pre-installation verification fails, it will report the following errors:

```
ERROR: The following base package image is bad: CSCOName
```

```
ERROR: Package verification failed : CSCOName aborting.
```

If you get this kind of error, you can be sure that there was a problem during package creation in the build. Once the installation has aborted, you can try the command on the command line, to find out why it failed

To verify a package after installation, run `pkgchk` as follows:

```
pkgchk -n name
```

Where *name* is the package name.

Package verification may fail after installation if the files:

- a. Were modified during postinstall. In this case, you may have to use the `installf` command during postinstall (for details, see the “[Installf](#)” section on page 21-99) to fix the problem.
- b. The files are volatile (that is, they change in size, as with configuration files). In this case, you may need to use `installif` to declare this file as volatile.

**Note**

When using `installf`, be sure not to use the `NMSROOT` variable in the file path. For example, if the path is `/opt/CSCOPx/examples/example1`, give this path explicitly, not as `$NMSROOT/examples/example1`. If you use `NMSROOT`, you will experience problems during custom path installation.

Solaris Getting Started Example

The following is an example of how to create an installable CD image of CWCS for a customer application. The *myapp* name refers to the application *my application*; it is truncated because Solaris package names must have five letters or less on Solaris (CSCOxxxx).

Before you use this example, verify that the tools are in the path.

```
# which sh
```

The return value should display

```
/bin/sh
```

Perl V5.0 or higher is required.

```
# which perl
```

The return value should display

```
/auto/em_tools/sol/bin/perl
```

```
# perl -v
```

The return value should display

```
This is perl, version 5.005_03 built for sun4-solaris
```

```
Copyright 1987-1999, Larry Wall
```

For this example, it is assumed that:

CISCO CONFIDENTIAL

- The sample application is in two tar-files, myapp.cd.tar and myapp.runtime.tar, both in the current directory. The installation script untars these files automatically and places them in the following target directories: myapp.cd.tar into myapp\disk1 directory structure and myapp.runtime.tar into myapp\install and myapp\runtime structures. The directory myapp\disk1 contains the disk.toc file. The following disk.toc describes our sample application's table of contents.

```
[RELEASE]
NAME=CWCS with Test Application
VERSTR=1.0

[COMPONENTS]
TAGS=cwcs cmfwd cmfj2 CSCOMyapp
UNINSTALLABLE=cmfwd CSCOMyapp
VISIBLE=cwcs cmfwd CSCOMyapp
CHOICE=cwcs cmfwd CSCOMyapp
DEFAULT=ALL

[ADVANCED_CHOICE_1]
ADVANCED_CHOICE_1_CONDITION=cmfwd.1.0.0-1.9.99
ADVANCED_CHOICE_1_TYPE=EXCLUSIVE
ADVANCED_CHOICE_1_DEFAULT=4
ADVANCED_CHOICE_1_1_TEXT=CiscoWorks Common Services (CWCS) Base Desktop
ADVANCED_CHOICE_1_1_TAGS=cmfwd
ADVANCED_CHOICE_1_2_TEXT=CWCS (including Base Desktop)
ADVANCED_CHOICE_1_2_TAGS=cwcs
ADVANCED_CHOICE_1_3_TEXT=myapp Application
ADVANCED_CHOICE_1_3_TAGS=CSCOMyapp
ADVANCED_CHOICE_1_4_TEXT=myapp Application and CWCS
ADVANCED_CHOICE_1_4_TAGS=cwcs cmfwd CSCOMyapp

[ADVANCED_CHOICE_2]
ADVANCED_CHOICE_2_CONDITION=TRUE
ADVANCED_CHOICE_2_TYPE=EXCLUSIVE
ADVANCED_CHOICE_2_DEFAULT=3
ADVANCED_CHOICE_2_1_TEXT=CiscoWorks Common Services (CWCS) Base Desktop
ADVANCED_CHOICE_2_1_TAGS=cmfwd
ADVANCED_CHOICE_2_2_TEXT=CWCS (including Base Desktop)
ADVANCED_CHOICE_2_2_TAGS=cwcs
ADVANCED_CHOICE_2_3_TEXT=myapp Application and CWCS
ADVANCED_CHOICE_2_3_TAGS=cwcs cmfwd CSCOMyapp
```

For this toc file you need the following:

- REGISTRY_ROOT provides the key name for component information. The value has been set for the CWCS release and has to be the same to keep compatibility with CWCS.



Note The NAME parameter in the [RELEASE] section is used to make a name of an executable. All spaces are replaced by underscores, but parenthesis, comma slashes, and other characters are not allowed in filenames.

The ADVANCED_CHOICE section determines what is the correct scenario depending on what is already installed on the target machine. In the case where the target machine has a previous version of CiscoWorks Common Services (CWCS) Base Desktop already installed, the customer has the following options:

- Upgrade to a new version of CiscoWorks Base Desktop
- Install CWCS 3.0
- Install the myapp application on top of CiscoWorks Desktop, or

CISCO CONFIDENTIAL

- Install the application bundled with CMF 3.0.

In the case of a fresh installation, you do not have the option of a separate myapp installation. In this case, there are only the remaining three options mentioned in the `ADVANCED_CHOICE_2` section.

- The directory `myapp/install` contains the `myapp.bprops` and `mypp.pkgpr` files. The directory `myapp/runtime` contains the subdirectories “`cgi-bin`” and “`htdocs`”, corresponding to the structure of similar directories of CWCS to be used by the web server. Substitute the two Perl scripts, `cgi-bin/myapp/myappcgi.pl` and `htdocs/myapp/myappframe.html`, with your application files. The following files are for linking our HTML and Perl files to the appropriate tree structure of the CWCS main web page:

- `htdocs\Xml\System\maintree\myapp.xml`
- `htdocs\Xml\System\maintree\myappcgi.xml`

Each developer must swap their XML files in their place. For details on integrating your application with CiscoWorks, refer to the “[Integrating Your Application with CWHP](#)” section on page 7-6.

- `pkgtools.runtime.tar` file is in `protopackages` directory of the SDK kit.
- `BuildImage` file is in the root directory of the SDK kit.
- `makesolpkg` file is in the root directory of the SDK kit.

To create a CWCS image, use the `sample1.sh` in the current directory.

- Launch `sample1.sh` with four arguments:

```
./sample1.sh arg1 arg2 arg3 arg4
```

where:

- `arg1` is the target directory;
- `arg2` is the directory with files `myapp.cd.tar` and `myapp.runtime.tar` (actually current directory)
- `arg3` is the directory with `buildImage` file
- `arg4` is the directory with `pkgtools.runtime.tar` file, `protopackages` tar files, `cab-files`.

The `sample1` file is shown below:

```
#!/bin/sh
#####
# INPUT:
# 1 - Target_dir
# 2 - myapp_dir
# 3 - buildImage_dir
# 4 - proto_dir pkgtools.runtime.tar and protopackages dir
# you should use to execute this script
#####

if [ $# -ne 4 ]; then
    echo "ERROR:sample2.sh called with insufficient args."
else
    Target_dir=$1
    myapp_dir=$2
    buildImage_dir=$3
    proto_dir=$4

    rm -rf $Target_dir
    mkdir $Target_dir

    /usr/local/bin/perl $buildImage_dir/buildImage -r -d $Target_dir \
    $proto_dir/pkgtools.runtime.tar \
    $myapp_dir/myapp.runtime.tar $myapp_dir/myapp.cd.tar \
    $proto_dir/cam.runtime.tar $proto_dir/cmfc.runtime.tar \
```

CISCO CONFIDENTIAL

```

$proto_dir/cmfv2.runtime.tar $proto_dir/cmfvd.runtime.tar \
$proto_dir/db.runtime.tar $proto_dir/dmgt.runtime.tar \
$proto_dir/eds.runtime.tar $proto_dir/ess.runtime.tar \
$proto_dir/grid.runtime.tar $proto_dir/jawt.runtime.tar \
$proto_dir/jchart.runtime.tar $proto_dir/jext.runtime.tar \
$proto_dir/jgl.runtime.tar $proto_dir/jpwr.runtime.tar \
$proto_dir/jre2.runtime.tar $proto_dir/jrm.runtime.tar \
$proto_dir/logmsg.runtime.tar $proto_dir/lotusxsl.runtime.tar \
$proto_dir/nmcs.runtime.tar $proto_dir/perl.runtime.tar \
$proto_dir/plugin.runtime.tar $proto_dir/pxhlp.runtime.tar \
$proto_dir/snmp.runtime.tar $proto_dir/swng2.runtime.tar \
$proto_dir/jext.runtime.tar $proto_dir/vorb.runtime.tar \
$proto_dir/web.runtime.tar $proto_dir/xml4j.runtime.tar \
$proto_dir/xrts.runtime.tar $proto_dir/eds.cab.tar
$proto_dir/jgl.cab.tar $proto_dir/swng2.cab.tar
$proto_dir/vorb.cab.tar

```

```
fi
```

The output of this command is very lengthy. The actual set of tar files may be different from this example. Here is the result:

```

#cd <Target Directory>
#ls
autoinstall.sh disk1 extract

```

- Now the necessary product (CiscoWorks Base Desktop, CiscoWorks, customer's application see the options in ADVANCED_CHOICE section above) can be installed by running setup.sh from the ./disk1 directory.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 22

Using the Java Plug-in

Java Plug-in (JPI) technology is a Sun Microsystems product that allows Java2 applets to run in web browsers.

All applets have been removed beginning with this version of CWCS. CiscoWorks Common Services does not require or depend on a browser VM or JPI technology to provide its services. However, the JPI is included in the shipping version of CWCS so that applications who do depend on the JPI can deploy their applets. CWCS includes the extractable EXE file for Windows, and a compressed TAR file and customized installation script (pam.sh) for Solaris. The software packages are CSCOPlug on Solaris and plug on Windows; at runtime, the JPI installables are located in *NMSROOT*/htdocs/plugin.

Applications can evaluate whether to use the Java Plug-in depending on their needs. The CWCS JPI team recommends that developers:

1. Determine if your application's features really require the JPI.
2. If your application does need it, use only the JPI version supplied with CWCS, and ensure your application works with that version of the JPI.
3. Do not hard-code the Plug-in version in your JSPs. Instead, pick the JPI version dynamically, as explained in the [“Using the Java Plug-in API”](#) section on page 22-2.

The following topics describe the Java Plug-in support in CWCS and how to use it with your application:

- [About the Java Plug-in Requirements](#)
- [Using the Java Plug-in API](#)
- [Accessing the JPI Configuration from CCR](#)
- [Using Tags Java Plug-in](#)
- [Using Client Local Resources](#)
- [JPI Technology References](#)

About the Java Plug-in Requirements

The Java Plug-in requires specific operating systems, browsers, and patches to run successfully. Both Solaris and Windows client machines need to have approximately 12 MB disk space for downloading and installing the Plug-in. On Windows, the C: drive needs to have approximately 10 MB.



Caution

It is recommended to uninstall any other versions of the Java Plug-in from the client machine before installing the CWCS- supported version.

CISCO CONFIDENTIAL**Table 22-1** Java Plug-in Requirements

Client OS/Browsers	JPI Version	System Requirements
Solaris 2.8/Netscape 7.0, Mozilla 1.7.13	1.4.2_10	To obtain patches, see the SunSolve support website . You will find patch clusters for Solaris operating system platforms J2SE Solaris 9 and J2SE Solaris 8.
Solaris 2.9/Netscape 7.0, Mozilla 1.7.13		
Windows 2000/Netscape 7.1/7.2, Microsoft Internet Explorer 6.0 SP2, Mozilla 1.7.13	1.4.2_10	No special requirements

Using the Java Plug-in API

The Java plug-in API is available to all applications. This API dynamically substitutes the values of the plug-in versions used by CWCS clients for Windows and Solaris platforms.

You can use the API to retrieve the Plug-in values by sending the operating system name, as follows:

- Windows: WIN-IE, WIN-NS
- Solaris: SOL

For Windows alone, WIN-IE returns the class ID for CWCS Internet Explorer clients and WIN-NS returns the version to be used by CWCS Netscape clients.

For example:

1. In a JSP file, include `com.cisco.nm.cmf.ssl.GetPluginVersion`.
2. `%=GetPluginVersion.getPluginVersion("WIN-IE")%` can be used in places where the class ID for the Java Plug-in is hard-coded.
3. `%=GetPluginVersion.getPluginVersion("WIN-NS")%` can be used in places where the type of EMBED tag for Java Plug-in is hard-coded.
4. `%=GetPluginVersion.getPluginVersion("SOL")%` can be used in places where the type of EMBED tag for Java Plug-in is hard-coded.
5. `%=GetPluginVersion.getPluginVersion("IE_PLUGINS PAGE")%` points to the URL from which the Plug-in can be downloaded.

You can view these values in the file `NMSROOT/lib/classpath/javaplugin.properties`.

Accessing the JPI Configuration from CCR

You can retrieve the values of the JPI configuration parameters from the Core Client Registry. CCR will contain a separate entry for each configuration parameter, as follows:

```
MDC (Application) Name: Core
Resource Type: Custom
Resource Name: WIN_IE_VERSION

MDC (Application) Name: Core
Resource Type: Custom
Resource Name: WIN_NS_VERSION
```

CISCO CONFIDENTIAL

```
MDC (Application) Name: Core
Resource Type: Custom
Resource Name: SOL_VERSION
```

```
MDC (Application) Name: Core
Resource Type: Custom
Resource Name: JavaPluginExe
```

The CCR resources WIN_IE_VERSION, WIN_NS_VERSION, SOL_VERSION, and JavapluginExe correspond to the WIN_IE, WIN_NS, SOL, and IE_PLUGINSPAGE parameters discussed in the “Using the Java Plug-in API” section on page 22-2.

To retrieve a JPI configuration parameter from CCR, use code like the following:

```
CCREntry ccrentry = new CCREntry("Core", "Custom", "Custom", "", "", "WIN_IE_VERSION");
CCRResponse ccrresponse = ccrinterface.retrieveEntry(ccrentry);
ccrentry = (CCREntry)ccrresponse.getReturnedValues().elementAt(0);
String resourceData = ccrentry.getResourceData();
```

You can query WIN_NS_VERSION, SOL_VERSION, and JavaPluginExe in the same way.

Using Tags Java Plug-in

To use the Java Plug-in, you must change your HTML pages to use Sun's JRE via the Java Plug-in software. Sun provides an [HTML converter tool](#) with the Sun JPI SDK. This tool automatically makes the necessary changes to the HTML web pages. For more information about the HTML converter tool, see http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/html_converter_more.html.

The tagging structure that the JPI requires, including the correct OBJECT and EMBED tags, is explained in the Sun Java Developer Guide chapter “Using OBJECT, EMBED and APPLET Tags in Java Plug-in” (see http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/using_tags.html).

Developers must ensure that the correct HTML tags have been used for each browser (EMBED for Netscape and OBJECT tag for Microsoft Internet Explorer). Otherwise, the Java Plug-in is not used and the applet is started using the browser's default Java Virtual Machine. After downloading and installing the Java Plug-in software from the CWCS Server, the applets would continue to execute under the Java Plug-in for any pages that are converted to use the Java Plug-in.

Using Client Local Resources

To allow each client to use its local resources (such as printing and cut/copy/paste), you must use the clientservices13.jar file, as follows:

- The clientservices13.jar file is located in *NMSROOT*/www/classpath. Applications need to download this version of the clientservices13.jar with the applets that need access to local resources.
- Applications may need to sign the application JAR file. Contact the Release Engineering team for details on doing this.

CISCO CONFIDENTIAL

JPI Technology References

The following external references may assist your understanding of Sun's implementation of the Java Plug-in.

- Sun main plug-in page: <http://java.sun.com/products/plugin/>
- Technical documentation: <http://java.sun.com/j2se/1.4.2/docs/guide/plugin/index.html>
- Applet caching:
http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/applet_caching.html
- Standard extension versioning: <http://java.sun.com/products/plugin/versions.html>
- Related jar indexing:
http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/jar_indexing.html



CISCO CONFIDENTIAL

CHAPTER 23

Using the Diagnostic and Support Utilities

CWCS provides diagnostic and customer-support utilities that:

- Help customers collect data about their CWCS installations and the applications installed with them.
- Give Cisco developers and technical support staff the information they need to resolve customer problems more quickly.

The following topics describe these utilities:

See this Topic	For Information on this Utility
Using Collect Server Info	Collect Server Info: Gathers all-in-one information about the state of the CWCS Server.
Using the MDC Support Utility	MDCSupport: Collects diagnostic output into a zip file.
Using SNMP Set and Walk	SNMP Set and Walk: SNMPwalks a device to troubleshoot or gather information about that device.
Using Packet Capture	Packet Capture: Captures live data from the CiscoWorks Server.
Using Logrot	Logrot: Rotates log files.

Using Collect Server Info

The Collect Server Info utility allows customers to quickly collect all-in-one information about the state of their CWCS Servers. This module is implemented as a Perl script. This script uses all OS-specific commands to get system information, process information and memory information. The following topics describe how to use this utility:

- [What Data Does Collect Server Info Gather?](#)
- [Customizing Collect Server Info](#)
- [Running CollectServerInfo](#)

What Data Does Collect Server Info Gather?

Collect Server Info collects the following information:

- System identity, including name, type, network data.
- System configuration. This includes:

CISCO CONFIDENTIAL

- Network settings
- CWCS Server name
- Shared secret
- TIBCO RVRD IP address
- Root account enable/disable
- Notification settings (where CWCS sends its notifications)
- Cluster management
- SMTP server name
- NTP configuration
- Security configuration
- Backup history configuration
- Application settings
- Software installed on the system, with detailed information about packages.
- Web server configuration and status.
- Servlet engine configuration and status.
- Database status and configuration with the sizes of database files.
- Process status and service status log files. If log files are very big, only the last portions could be collected (for example, the last 300 records).

Customizing Collect Server Info

To add your application's troubleshooting information to the Collect Server Info utility:

Step 1 Create and add a shell script or perl file to the `$NMSROOT/collect` directory. This file will collect your application-specific information. For example, `myInfo.exe`.

Step 2 Add a text file named `myInfo.txt` containing the following lines to the `$NMSROOT/collect` directory:

```
Name=My
Option=my
Descr=My Status
Order=A3
```

Running CollectServerInfo

To run `CollectServerInfo`:

Step 1 From the CWHP, select **Server > Diagnostics > Collect Server Info**. The Collect Server Information page displays.

Step 2 Click **Create** to view a list of options such as web server, operating system, and process status.

CISCO CONFIDENTIAL

Step 3 Select the options by clicking in the appropriate boxes, then **Finish**. To collect your application-specific information, click on the checkbox for MY Information. If MY Information does not appear, perform the steps in the “[Customizing Collect Server Info](#)” section on page 23-2.

The Collect Server Information page refreshes with a link to the server information report.

Step 4 Click on the hyperlink to view the report.

Using the MDC Support Utility

The MDC support utility is a console application written in C++. It allows users enter an optional command line argument (the output path/location of where the zip file should be located) to collect diagnostic output into a zip file.

The resulting zip file consists of SQL DB files, Core Client Registry (CCR) files, configuration and log files of Apache and Tomcat, Core log file, WinNT event logs and System information like host details, memory, disk, and network details. The utility can also:

- Run other support utilities registered by applications with CCR.
- Register these other utilities with the CCR.

MDCSupport in Core provides the same information as the Collect Server Info utility, and also provides information by executing the application-specific executables registered with CCR.

The following topics describe how to use this utility:

- [About the MDC Support Utility Requirements](#)
- [What Data Does the MDC Utility Collect?](#)
- [Registering Alternative MDC Support Utilities](#)
- [Running MDC Support](#)

About the MDC Support Utility Requirements

There are no disk space requirements for MDC Support other than the room required for the zip file.

What Data Does the MDC Utility Collect?

The MDC Support utility collects the following information:

- SQL database files under <CoreRoot>\Sybase\Db*
- CCR file <CoreRoot>\etc\regdaemon.xml
- Schema files under <CoreRoot>\etc
- Apache (web server) configuration and log files located at:
 - <CoreRoot>\apache\conf*
 - <CoreRoot>\apache\log*
- Tomcat configuration and log files (the servlet container technology standard at Cisco, currently deployed on the CCI environment), and the XML file located at:

CISCO CONFIDENTIAL

- <CoreRoot>\tomcat\conf*
- <CoreRoot>\tomcat\log*
- <CoreRoot>\tomcat\mdc\web-inf\web.xml
- <CoreRoot>*.log: Log file created when Tomcat crashes, containing a stack trace.
- Log files registered with CCR for Core under <CoreRoot>\MDC\log, including the audit and operation logs
- Installation log files of the form cw*.log found in the System drive
- Registry subtree: [HKEY_LOCAL_MACHINE][SOFTWARE][Cisco][MDC]
- Windows NT event logs:
 - System event log file
 - Application event log file
- Host environment information:
 - OS version/NT service packs
 - Physical RAM
 - Disk Space on all volumes
 - Computer Name
 - Virtual Memory size

**Note**

The MDCSupport utility also queries CCR for any other, alternative support utilities and runs them automatically if they are registered. For more information, see the “[Registering Alternative MDC Support Utilities](#)” section on page 23-4.

Registering Alternative MDC Support Utilities

You can have the MDC Support utility automatically run other, alternative MDC support utilities and include their output in the *MDCSupportInformation.zip* file. To do this, register the alternative support utility with the Core Client Registry using the following command:

Name	ccraccess
Runtime Location	The location of the support utility.
Syntax	<pre>ccraccess -addMDC mdcName mdc_root_dir ccraccess -addResource mdcName Custom Custom support_utility_path EMPTYSTRING MDCSupportExecutable</pre>
Arguments	<p>The utility registered takes two arguments:</p> <ul style="list-style-type: none"> • The first argument is the core installation directory (usually \$NMSROOT/MDC). • The second argument is the directory in which the MDC’s support utility needs to add all the data it wants zipped by the MDCSupport utility. <p>Tip To make it easier to identify the files coming from each MDC, it is highly recommended that each MDC create a temporary directory inside this directory and put all the files there.</p>

CISCO CONFIDENTIAL

Name	ccraccess
Example	<p>If SampleMDC is installed at C:\apps\MDC\SampleMDC:</p> <ol style="list-style-type: none"> Register MDC with CCR: <pre>ccraccess -addMDC SampleMDC C:\apps\MDC\SampleMDC</pre> Register support utility with CCR: <pre>ccraccess -addResource SampleMDC Custom Custom C:\apps\MDC\SampleMDC\bin\SampleSupport.exe EMPTYSTRING MDCSupportExecutable</pre>

When MDCSupport is run (with no arguments):

- MDCSupport creates a temporary directory: <CoreRoot>\etc\mdcsupporttemp
- MDCSupport collects all requested data
- MDCSupport queries CCR and finds that SampleMDC has a utility registered, and creates a process: C:\apps\MDC\SampleMDC\bin\SampleMDCSupport.exe and passes it the two arguments <CoreRoot> and <CoreRoot>\etc\mdcsupporttemp. (Note that the two arguments are full paths).
- SampleMDCSupport creates a SampleMDC directory under mdcsupporttemp dir, and adds all the files it needs to this directory.
- MDCSupport zips the contents of mdcsupporttemp and then deletes mdcsupporttemp and all of its contents.

Running MDC Support

The *MDCSupport.exe* utility is a console application that takes one optional command-line argument: the output path and location where the resulting zip file should be created. If you do not specify a value for this argument, the output is created in <CoreRoot>\etc.

The output consists of an *MDCSupportInformation.zip* file. An *MDCSupport.log* text file in the zip package acts as a log of what has been collected and notes any errors encountered. Other information in the file includes:

- The time the support tool was run.
- The Core version installed.
- The installation location of Core.
- The System path.
- Host environment information.
- Other MDCs support utilities executed.

To run MDCSupport.exe:

-
- Step 1** If there is enough space on the disk where the MDC suite is installed, enter the following on the command line (with no arguments):

MDCSupport.exe

The file *MDCSupportInformation.zip* file is created in the <CoreRoot>\etc directory.

CISCO CONFIDENTIAL

- Step 2** If you use another disk to store the temporary files and the output, enter the output path and location where the resulting zip file should be created. For example:

MDCSupport D:\temp

The file *MDCSupportInformation.zip* is created in the D:\temp directory.

Using SNMP Set and Walk

Use SNMP Set and Walk to troubleshoot or gather information about a device. The following topics describe how to use this utility:

- [About the SNMP Set and Walk Requirements](#)
- [Running SNMP Set and Walk](#)
- [Updating the MIBs for SNMP Walk](#)



Note

This tool is intended for customers running CiscoWorks and is designed to integrate with CWCS.

About the SNMP Set and Walk Requirements

- Supported versions: CiscoWorks CD One, CD One 2nd edition, CD One 2nd Edition Patch 1, CD One 3rd Edition, CD One 4th Edition, CD One 5th Edition, or CiscoWorks Common Services 2.2 and 3.0.
- Supported platforms:
 - Windows2000 Server with SP2 (Pentium III & 4)
 - Windows2000 Advanced Server (not configured as either a Domain Controller or a Terminal Server) (Pentium 4)
 - Sun Solaris 7 and 8 (2.7 and 2.8) (UltraSPARC II, Ili, IJe, III, and IIIc)
- All prerequisites for CiscoWorks must be met for both client and server.

Running SNMP Set and Walk

SNMP Set and Walk fetches credentials from the CWCS Device Credentials Repository (DCR), if available. Otherwise, it assumes the default credentials for SNMP v1/v2c.

- Step 1** Using your web browser, log in to CiscoWorks as local administrator.

- Step 2** You must launch the SNMP Set and Walk tools from the Device Center:

To launch SNMP Set:

- a. Select **CiscoWorks HomePage > Device Trouble Shooting > Device Center**.
- b. Enter an IP address or select one from the Device Selector list.
- c. Select **Go**. The Tools column shows the list of available tools.

CISCO CONFIDENTIAL

d. Select **SNMP Set**.

To launch SNMP Walk:

- a. Select **CiscoWorks HomePage > Device Trouble Shooting > Device Center**.
- b. Enter an IP address or select one from the Device Selector list.
- c. Select **Go**. The Tools column shows the list of available tools.
- d. Select **SNMP Walk**.

Step 3 Enter the following information:

Field	Description
Object ID, Instance ID or number	Either an IP address or a hostname. If it is a hostname, the hostname <i>must</i> resolve correctly either through DNS, local hosts file, or NIS. Note WINS has not been tested and is <i>not</i> supported. Note The device does not need to be in RME's inventory or discovered on the Campus Manager topology map.
community string	This can be either a read-only community string, a read-write string, or a read-write-all string. Remember, it is case-sensitive! Note Do not use this field if you are using SNMPv3.
SNMPv3 user, password, and authentication protocol	For SNMPv3 users only. If you are not using SNMPv3, leave these fields blank. Note SNMPv3 support is for authNoPriv only at this time.
starting object ID	Optional: The point in the MIB tree where you want to start the walk. If this field is left blank, SNMP Walk will start the walk from: <ul style="list-style-type: none"> • For SNMP Set, you can specify multiple OIDs. • For SNMP Walk, you can enter only one OID.
translate results	Optional: If you check this box, all OIDs will be shown numerically instead of translated. This can help troubleshoot issues that require the sysObjectID.
SNMP timeout	Optional: The SNMP timeout (default = 10 seconds).
version	The version of SNMP to use. Anything that is a 64-bit counter must be queried with v2c or v3.

Step 4 To run SNMP Walk, select **Ok**. SNMP Walk will start the walk based on the parameters you entered. A full walk may take a *long* time, so be patient.

SNMP Set requires two additional fields:

Field	Description
object type	The type of object to be set. You can select the object type from the drop-down list.
new value	The new value to be set for the object.

CISCO CONFIDENTIAL

To set multiple objects, fill in all the required fields, then click **Next**. Repeat this until you are ready to send the SETs to the device. At this point, each SET goes into its own packet.

Updating the MIBs for SNMP Walk

By default, SNMP Walk uses the MIBs found in the installed nmldb.jar. This jar file is found in `$NMSROOT/nmim/nmldb`.

If a newer nmldb jar file is downloaded from CCO, you can run the `$NMSROOT/bin/upgradeJTMibs.pl` script to add those MIBs to SNMP Walk.

To run the script:

- On Windows platforms, enter:

```
C:\> $NMSROOT\bin\perl $NMSROOT\bin\updateJTMibs.pl <path to nmldb.zip>
```

(where `$NMSROOT` is the path where CiscoWorks is installed)

For example:

```
C:\> C:\Progra~1\CSCOpX\bin\perl C:\Progra~1\CSCOpX\bin\updateJTMibs.pl
C:\nmldb.1.0.025.zip
```

- On UNIX platforms, enter:

```
# /opt/CSCOpX/bin/updateJTMibs.pl <path to nmldb.zip>
```

For example:

```
/opt/CSCOpX/bin/updateJTMibs.pl /tmp/nmldb.1.0.025.zip
```

Using Packet Capture

Use the Packet Capture utility to capture live data from the CiscoWorks machine. This utility is intended for customers running CiscoWorks and is designed to integrate with CWCS.



Note Packet Capture is a troubleshooting tool. It should *not* be used as a general-purpose sniffer.

The following topics describe how to use this utility:

- [About the Packet Capture Utility Requirements](#)
- [Running Packet Capture](#)

About the Packet Capture Utility Requirements

- Supported versions: CiscoWorks CD One, CD One 2nd edition, CD One 2nd Edition Patch 1, CD One 3rd Edition, CD One 4th Edition, CD One 5th Edition, Common Services 2.2 and 3.0

CISCO CONFIDENTIAL

- Supported platforms:
 - Windows2000 Server with SP2 (Pentium III & 4)
 - Windows2000 Advanced Server (not configured as either a Domain Controller or a Terminal Server) (Pentium 4)
 - Sun Solaris 7 and 8 (2.7 and 2.8) (UltraSPARC II, Ili, IJe, III, and IIIc)
- All prerequisites for CiscoWorks must be met for both client and server.

Running Packet Capture

To run the Packet Capture utility:

-
- Step 1** If you are using one of the supported Windows platforms: Install winpcap.exe in NMSROOT/objects/jet/bin directory. The version of winpcap.exe is 3.0 and is intended for use with Windows only.
- Step 2** Using your web browser, log in to CiscoWorks as Admin.
- Step 3** To launch Packet Capture:
- a. Select **CiscoWorks HomePage > Device Trouble Shooting > Device Center**.
 - b. Enter an IP address or select one from the Device Selector list.
 - c. Select **Go**. The Tools column shows the list of available tools.
 - d. Select **Packet Capture**.
- Step 4** The currently-archived capture files are displayed. (If no capture files have been archived, a message will indicate that there are no capture files.) From this screen, you can:
- Create a new capture
 - Delete an existing capture file
- Step 5** Select **Create** to configure which packets should be captured.
- a. If you have multiple interfaces on the machine, select the interface on which you wish to capture packets. The Address(es) field accepts one or more addresses (separated by a single space) to match when capturing.
 - b. You can configure the protocol and port on which to capture (the default), or you can select from the pre-configured list of common CiscoWorks applications.
 - Protocols and ports: Select the protocols (TCP, UDP, or ICMP) you would like included in the capture. Then enter the list of ports to capture on for TCP and UDP. The Port(s) field accepts one or more TCP or UDP ports (separated by a single space).
 - Pre-configured list of common CiscoWorks applications: Select **Application**, then select one or more applications from the list.
 - c. Specify when to stop the packet capture. You can choose to terminate the capture:
 - After a set amount of time. By default, the capture stops after 60 seconds (1 minute).
 - After the filter has captured a certain amount of data.
 - After a certain number of packets have been captured.
- Step 6** While the capture is running, an applet will update the number of packets captured and capture size in real time. To stop the capture before the auto-stop condition is met, click **Stop**.

CISCO CONFIDENTIAL

- Step 7** Each capture is saved on the server in the *NMSROOT*/htdocs/jet directory. The files are created in binary libpcap format with a .jet extension. You can use your web browser to download these files, then email them to the TAC for further analysis.
-

Remarks

- On Solaris platforms, Packet Capture installs a setuid root binary in /opt/CSCOpX/objects/jet/bin. This binary is only executable by users in the casusers group, but if this is still too risky, consider revoking its setuid privileges until you need to use it.
- Packet Capture is only accessible to CiscoWorks users that have System Administrator (admin) privileges. It is not recommended that you change this.

Using Logrot

The logrot utility is a log rotation program designed for use with CiscoWorks. While it depends on CiscoWorks being installed on the same machine, logrot is not limited to rotating only CiscoWorks log files. You can use logrot to rotate any file you wish.

The logrot utility has some unique advantages over other log rotation programs:

- It can rotate logs while CiscoWorks is running or it can shut down CiscoWorks before rotating the logs.
- It can optionally archive and compress rotated logs.
- It can be configured to rotate logs only when they have reached a certain size.
- It has a built-in configurator that makes adding new files very easy.

The following topics describe how to use Logrot:

- [Configuring Logrot](#)
- [Running Logrot](#)
- [Using Logrot Command Line Switches](#)
- [Troubleshooting Logrot](#)

Configuring Logrot

To configure Logrot:

- Step 1** Navigate to the logrot directory:
- On Windows platforms, enter:

```
C:\> NMSROOT\bin\perl.exe NMSROOT\bin\logrot.pl -c
```
 - On UNIX platforms, enter:

```
# /opt/CSCOpX/bin/logrot.pl -c
```
- Step 2** Select **Edit Variables** and enter the following information:

CISCO CONFIDENTIAL

Variable	Description
backup directory	The backup directory must already exist. If you do not set a backup directory, then each log will be rotated in its current directory.
restart delay	Optional: Daemon Manager restart delay.

Step 3 Return to the main menu, then select **Edit Log Files**.

Step 4 Enter the following information:

Field	Description
log file location	Enter the full path to the logfile. If the full path is not entered, the default logfile path for your operating system is prepended (for example, /var/adm/CSCOpX/log on UNIX). Tip To add multiple logfiles at once, use '*' in the file name. The '*' character matches zero or more characters in a filename. Note You must use DOS file names when specifying the logfile path on Windows.
archives to keep	Enter the number of archive revisions to keep. If you don't want to keep any archives, enter 0 for this option.
file size	Enter the maximum file size in kilobytes (KB). The log will not be rotated until this size is reached.
compression option	Allowable options are: Z—The archived file will be compressed using UNIX compress(1). bz2—The archived file will be compressed using bzip2 (available by default on Solaris 8 and above only). gz—The archived file will be compressed with GNU gzip (available by default on Windows only). blank—On Windows platforms, the only valid values are to leave this option blank or set it to gz.
restart delay	How long to wait (in seconds) before proceeding after the Daemon Manager is shut down. This option is only used if logrot is run from the command line using the -s switch (see the “Using Logrot Command Line Switches” section on page 23-12). The default delay is 60 seconds.

Running Logrot

Logrot is typically run as a UNIX cron or Windows AT job.

Step 1 Before automating logrot, verify that it runs on-demand:

- To run logrot on UNIX platforms, enter:

CISCO CONFIDENTIAL

```
/opt/CSCOpX/bin/logrot.pl
```

- To run logrot on Windows platforms, enter:

```
NMSROOT\bin\perl.exe NMSROOT\bin\logrot.pl
```

Step 2 The following commands will run logrot every day at 1:00 AM. The UNIX cron line will also send all output of the command to root via email.

- To set up cron on UNIX, enter:

```
0 1 * * * /opt/CSCOpX/bin/logrot.pl 2>&1 | /usr/lib/sendmail root
```

- To set up AT on Windows, enter (all on one line):

```
at 01:00 /every:M,T,W,Th,F,S,Su C:\progra~1\CSCOpX\bin\perl.exe  
C:\progra~1\CSCOpX\bin\logrot.pl
```

This assumes CiscoWorks is installed in the default location on Windows.

Using Logrot Command Line Switches

The logrot utility accepts the following command-line switches:

Switch	Description
-v	Output verbose messages.
-s	Shut down dmgt (the Daemon Manager) before rotating the logs. This can be a safer way of performing log rotations (see the “Known Problems with Logrot” section on page 23-13). Note To specify the restart delay, see the “Configuring Logrot” section on page 23-10 .
-c	Re-run the configurator. This option can be specified at any time.

Troubleshooting Logrot

The following topics can help you troubleshoot the logrot utility:

- [Verifying Files and Time Cycles](#)
- [Verifying Scheduled Tasks](#)
- [Viewing the Scheduled Jobs Log File](#)
- [Verifying Logrot Status](#)
- [Known Problems with Logrot](#)

Verifying Files and Time Cycles

You can run the **at** command from the command prompt to see if the files specified and the time cycles are correct. For example:

```
C:\Documents and Settings\Administrator>at
```


CISCO CONFIDENTIAL

will produce output similar to this:

Status ID	Day	Time	Command Line
13	Each M T W Th F S Su	8:22 PM	C:\progra~1\CSCOpX\bin\perl.exe C:\progra~1\CSCOpX\bin\logrot

Verifying Scheduled Tasks

To verify scheduled tasks on the server, select **Start > Settings > Control Panel > Scheduled Tasks**. There should be an AT job present, AT<ID> (for example, AT13), that can be verified for correctness.

Viewing the Scheduled Jobs Log File

To look at the scheduled jobs log file:

1. Select **Start > Settings > ControlPanel > Scheduled Tasks**.
2. Select the AT job.
3. Select **Advanced > View Log**.

Verifying Logrot Status

To see if logrot is failing, run it from the command line using the -v (verbose) flag.

Known Problems with Logrot

After shutting down Daemon Manager when rotating daemons.log online, you may find that the daemons.log fills up with a lot of NUL (^@) characters. This is a side-effect of online log rotation. Daemon Manager should archive this file for you when it restarts. This does not cause any application problems, and can be worked around by doing an offline rotation of daemons.log.

When reporting problems, please include the verbose output of logrot (use the -v switch) as well as the logrot.conf file. The logrot.conf file can be found in /opt/CSCOpX/objects/logrot on UNIX and in *NMSROOT*\objects\logrot on Windows.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 24

Using SNMP Services

CWCS provides support for SNMPv1, SNMPv2c, and SNMPv3.

SNMPv3 support is new in this version of CWCS. SNMP v3 support enhances the security of SNMP operation over the existing support for the SNMPv1/v2c model. It provides the degree of authentication and privacy required to perform network management operations securely.

CWCS SNMPv3 support allows you to:

- Address threats like information modification, masquerade, and disclosure and message stream modification.
- Do SNMP requests using SNMPv3.
- Automatically discover SNMP engine parameters.
- Get and Set SNMPv3 engine parameters.
- Handle SNMPv3-related error conditions.
- Set the number of outstanding requests.
- Automatically re-localize keys.
- Use existing support for SNMPv1/SNMPv2c.

The following topics describe how to use CWCS SNMP Services with your application:

- [Why SNMPv3?](#)
- [How SNMP Support Works](#)
- [Using CWCS SNMP Services](#)

For basic information on CWCS SNMP Services, see the “[About SNMP Service Components](#)” section on page 6-15.

For more information about CWCS SNMP Services, see:

- *SNMPOnJava: Changes for SNMPv3 (authNoPriv)DS: EDCS-309325*

Why SNMPv3?

SNMPv3 is included in this release of CWCS to address threats not addressed in the existing SNMPv1/v2c model:

- **Information Modification:** An entity can alter an in-transit message generated by an authorized entity in such a way as to effect unauthorized management operations, including the setting of object values.

CISCO CONFIDENTIAL

- **Masquerade:** Management operations not authorized for some user may be attempted by assuming the identity of an authorized user.
- **Disclosure:** An entity can eavesdrop on the exchanges between managed agents and a management station and thereby learn the values of managed objects or learn of trap events.
- **Message Stream Modification:** The SNMP is designed to operate over a connection- less transport service, which may operate over any sub-network service. There is a threat that SNMP messages could be reordered, delayed, or duplicated to effect unauthorized management operations.

The SNMPv3 security model addresses the above threats in the following ways:

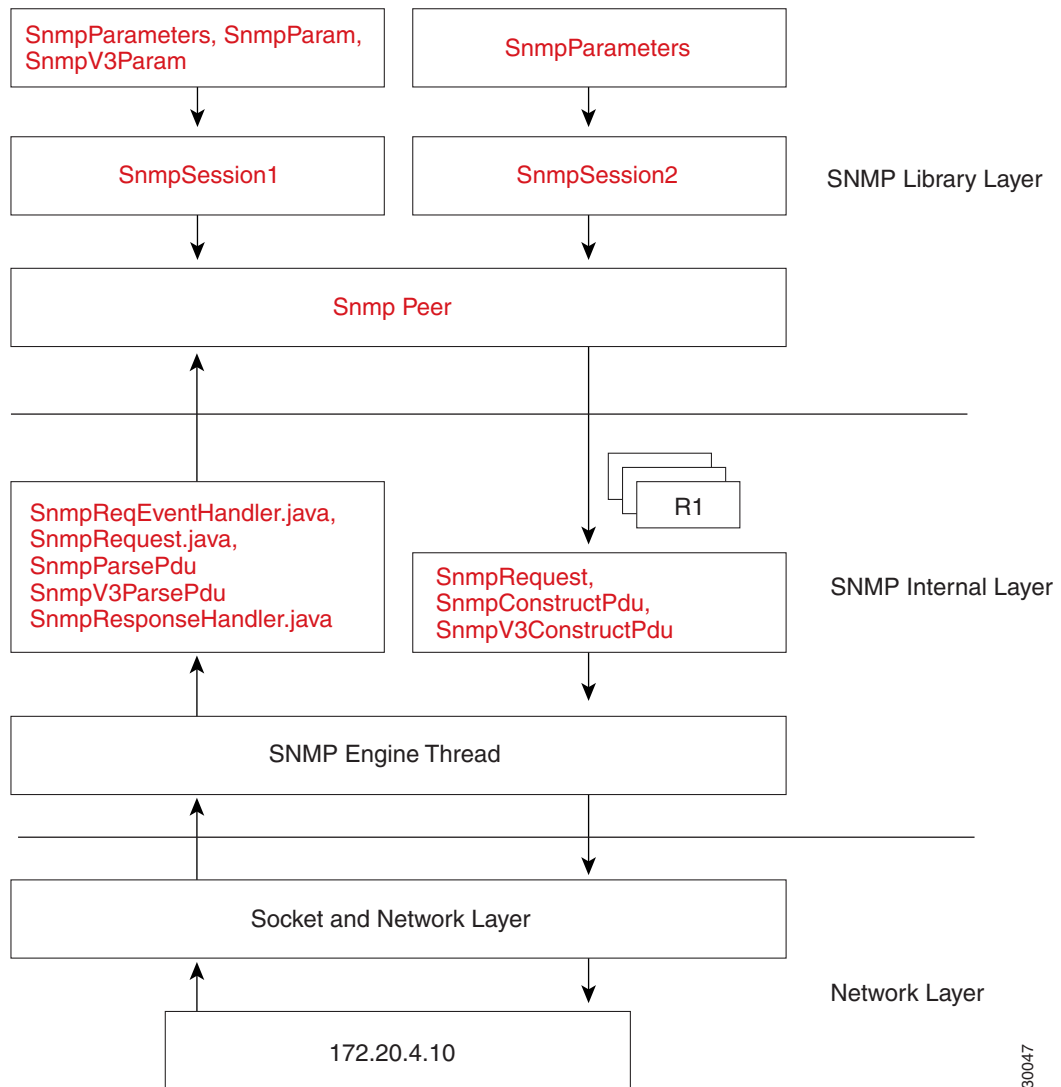
- Verify that each received SNMP message has not been modified during its transmission through the network.
- Verify the identity of the user who generates the SNMP requests.
- Detect received SNMP messages requesting or containing management information, whose time of generation was not recent.



Note For more information on the User-based Security Model (USM) for SNMPv3, refer to RFC 3414.

How SNMP Support Works

Figure 24-1 shows a high-level system flow for CWCS support of SNMP. The names of classes that were changed to handle SNMPv3 features are shown in red.

CISCO CONFIDENTIAL**Figure 24-1 CWCS Support for SNMP**

130047

Using CWCS SNMP Services

CWCS SNMPv3 allows the user to work in the authNoPriv mode of the SNMPv3 security model. This mode provides packet-level security, integrity protection, and replay protection. SNMPv3 support is enabled automatically by passing an SNMPv3 credential to the SNMPOnJava library. The flow of SNMPv3 is intermingled with that of SNMP v1/v2c.

CWCS SNMPv1/v2c/v3 support is provided in the SNMPOnJava library. This library provides a series of APIs for applications to use. The SNMPOnJava library is divided into two main sections:

- The main library: Contains the bulk of the main classes dealing with SNMP. For a summary of the classes in this library, see the [“About the SNMP Classes in the Main Library”](#) section on page 24-4.
- The futureapi: Contains credentials-oriented classes and future extensions. For a summary of the classes in this library, see the [“About the SNMP Classes in the Futureapi”](#) section on page 24-5.

CISCO CONFIDENTIAL

For details on each of the classes available in the *SNMPOnJava* library, see: <http://mspring-u10.cisco.com/cvw/MOJO/packages.html>.

The main features of CWCS support for SNMPv3 include:

- SNMPv3 is available for all applications.
- New APIs are available to get user credentials from applications.
- Applications can directly calculate the localized key from the user password.
- Applications can compute the local notion of an Agent's engine time.
- Automatic re-localization of keys.
- New APIs to expose the SNMPv3-engine-related parameters and localized keys to applications.
- Applications can pass SNMP-engine parameter information and localized keys to the library.
- Backward compatibility with the existing SNMP v1/v2c library.



Note SNMP engine parameters are `SnmEngineID`, `SnmEngineTime`, `SnmEngineBoots`, and local notion of Agent's time.

About the SNMP Classes in the Main Library

The main library contains the classes shown in [Table 24-1](#).

Table 24-1 *SNMPOnJava Classes: Main Library*

Class	Description
<code>SnmRequest</code>	Responsible for creating a SNMP request for one or more SNMP operations. In SNMPv3, these classes do SNMP engine parameter discovery like getting the <code>SnmEngineID</code> , <code>SnmEngineBoots</code> , and <code>SnmEngineTime</code> values from the device. They also provide authentication check in addition to error handling.
<code>SnmPeer</code>	Contains information about the peer, such as the IP address, port number, and parameters like maximum request packet size, maximum number of varbinds, and the permissible outstanding requests to which the SNMP call is to be sent. In SNMPv3, these classes hold the <code>SnmEngineID</code> .
<code>SnmParameters</code>	Contains information about credentials (community strings in case of v1/v2c) and the protocol version for an SNMP session. In SNMPv3, it holds SNMPv3 credential.
<code>SnmParam</code>	The base class for all credential-related classes.
<code>Snmv3Param</code>	Represents the SNMPv3 credential fields. It contains methods and constructors to get credential information from the user.
<code>SnmMain</code>	Initializes the SNMP environment, which contains default settings for every SNMP call. The initialization tasks include the number of threads for making SNMP calls, maximum retry, maximum timeout, protocol version, request packet size, retry policy, etc. In SNMPv3, this class holds a new property to specify the maximum number of outstanding requests for a peer.
<code>SnmReqEventHandler</code>	Handles the inherent asynchronous nature of <i>SNMPOnJava</i> . This class calls appropriate methods in other classes to process responses from the Agent. In SNMPv3, this class handles SNMPv3-specific conditions, such as unknown Engine ID, authenticity of the received message, etc. It checks the response and updates the SNMP Engine parameters.

CISCO CONFIDENTIAL**Table 24-1** *SNMPOnJava Classes: Main Library (continued)*

Class	Description
SnmpMultiplexReqDispatcher	Multiplexes one or more SNMP v1/v2c requests. It also prevents multiplexing SNMP v1/v2c and SNMPv3 requests.
Snmpv3ConstructPdu	Constructs the raw SNMPv3 message by translating the higher level information in SnmpRequest and SnmpParameters to the low-level byte stream information required by the device. It encodes the SNMPv3 message parameters – such as msgVersion, msgID, msgSecurityModel, SnmpEngineTime, SnmpEngineBoots, SnmpEngineId, msgAuthentication Parameters(digest), msgUserName, and others – to form a portion of the SNMPv3 message. Encoding the contextEngineName, contextEngineId, and the SNMPv2c PDU forms the remaining portion of the SNMPv3 message.
Snmpv3ParsePdu	Parses the raw SNMPv3 PDU received from the device for SNMPv3 message parameters, such as msgVersion, msgID, msgSecurityModel, SnmpEngineTime and SnmpEngineBoots, SnmpEngineId, msgAuthentication Parameters(digest), msgUserName, etc.
Snmpv3Param	Allows you to assign and fetch SNMPv3 credentials from applications. These include username, mode of SNMPv3 operation, authentication password, the context Engine ID and the context name.
SnmpResponseHandler	Receives the raw datagram object from the Java socket. It forwards the datagram to the appropriate PDU-handling classes, like SnmpV3ParsePdu and SnmpV2cParsePdu, for further processing of the received response.

About the SNMP Classes in the Futureapi

The futureapi library contains the classes shown in [Table 24-1](#).

Table 24-2 *SNMPOnJava Classes: Futureapi Library*

Class	Description
SnmpCommunityLoader	TheSnmpCommunityLoader class supports the file based SNMPv1/v2c and SNMPv3 credentials.
SnmpCommunity class	TheSnmpCommunity class stores v1/v2c and v3 credentials in its data structures.It also provides methods to assign and fetch the credentials.
SnmpPeerManager class	The SnmpPeerManager class allows you to create and fetch session along with the credential for a device. It also allows you to get the credential directly instead of getting it from theSnmpCommunity class.
SnmpCommand class	The SnmpCommand class allows users to pass their credentials during runtime.It also provides applications a better control to handle the SNMP call.
SnmpFuture class	TheSnmpFuture abstract class is the base class for all future objects handling SNMP operations. In SNMPv3, this class has been modified to handle SNMPv3-specific errors. The BouncyCastle library (a third-party library) is used to perform MD5/SHA-1 digest calculation.

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 25

Using NT Services

The CWCS NT Services (CSCSvc) are part of the CWCS Base Services, which include basic CWCS components necessary to support a web-based application. These components include the CWCS web server, CWCS security, the Tomcat servlet engine, and the JRE.

The CWCS NT Services contain NT support for communication services commonly available on UNIX. The services provided here are TFTP, RCP, TELNET and Syslog.

The following topics describe how to use CWCS NT Services with your application:

- [Understanding CWCS NT Services](#)
- [Using CWCS NT Services](#)

For basic information on CWCS NT Services, see the “[About NT Service Components](#)” section on page 6-15..

For more information about CWCS NT Services, see:

- *Mjollnir - CMF 2.3 System Functional Specification: (EDCS-283137)*
- *Mjollnir - CMF 2.3 PRD (EDCS-263430) I*
- *Config Transport Subsystem: System Functional Specification (ENG-16050)*
- *Rigel CMF: System Functional Specification (ENG-30118)*

Understanding CWCS NT Services

This topic covers basic information about the following CWCS NT Services

- [About the NT TFTP Service](#)
- [About the NT Telnet Service](#)
- [About the NT Service APIs](#)
- [About the NT RCP Service](#)
- [About the CRMLogger Service](#)

With this release of CWCS, NT Services has had the following changes:

- Limited remote access (for example, FTP, RCP, RSH) to the CWCS Server is provided to those users who are permitted to log in to the CWCS Server.
- The Syslog NT Service must support a continuous burst of at least 1,000 messages per second for one hour only. It should support at least 200 messages per second under normal conditions.

CISCO CONFIDENTIAL

- For the RSH service, the length of the rhosts and rusers keys should not exceed 2 KB. This is the recommendation given by Microsoft.

CWCS NT Services are base services, enabled by default; no explicit request for this bundle is required.

Network service bundles include System service bundles. If you enable Network services, all System service bundles are also enabled.

If your application needs a system or network service, then you must register the required service layer. For example, if your application needs the Sybase engine, then your application will need to register the system services layer of CWCS. This layer may contain many other services that you do not need, but you must register that layer to get any services from that layer (see the [“Registering for CWCS Services” section on page 5-4](#)).

About the NT TFTP Service

The TFTP transport mechanism is based on the SNMP set and get operations, using the SNMP library available with CWCS. TFTP can be used for catalyst devices and for IOS devices. TFTP cannot be used to get the startup configuration except under special circumstances (for some high-end devices for a particular setup). The transport subsystem uses the CWCS Tftpserver application. It obtains required environment information (including the location of the tftpboot directory and the location of the MIB data file for the MIB objects used in the SNMP operations) from the CWCS environment variables.

[Table 25-1](#) shows the set of MIB objects used for various transport operations.

Table 25-1 MIB Objects Used for TFTP

Operation	IOS Devices v10.3 & earlier	IOS Devices v11.3	Catalyst Devices v10.3 & earlier	Catalyst Devices v11.3
GetRunning Configuration	WriteNet/OLD-CISCO-MIB	COPY-COFIG-TABLE	TftpGrp/CISCO-STACK-MIB	TftpGrp/CISCO-STACK-MIB
GetStartupConfiguration	Same as RunningConfiguration	COPY-CONFIG-TABLE	Invalid	Invalid
UpdateRunning Configuration	NetConfigSet/OLD-CISCO-SYSTEM-MIB	COPY-CONFIG-TABLE	TftpGrp/CISCO-STACK-MIB	TftpGrp/CISCO-STACK-MIB
RunningToStartup Configuration	WriteMem/OLD-CISCO-SYSTEM-MIB	COPY-CONFIG-TABLE	Invalid	Invalid
Overwrite Startup Configuration	WriteMem/OLD-CISCO-SYSTEM-MIB	COPY-CONFIG-TABLE	Invalid	Invalid
Reload Device	TsMsgSend/OLD-CISCO-SYSTEM-MIB	COPY-CONFIG-TABLE	TftpGrp/CISCO-STACK-MIB	TftpGrp/CISCO-STACK-MIB

About the NT Telnet Service

The telnet transport mechanism creates the telnet session for the device using telnet passwords and enable passwords from the CWCS database. For the telnet service to work, your application inventory must have all the relevant access parameters. You can use the TELNET services for both for IOS and catalyst devices. [Table 25-2](#) shows the commands used to perform each transport operation.

CISCO CONFIDENTIAL**Table 25-2 Telnet Operations and IOS/Catalyst Commands**

Operations	IOS Devices	Catalyst Devices
GetRunning Configuration	write term	write term
GetStartupConfiguration	show config	Invalid
UpdateRunning Configuration	config term	Set commands
RunningToStartup Configuration	write mem	Invalid
Overwrite Startup Configuration	write erase write mem	Invalid
Reload Device	reload	reset

About the NT Service APIs

CWCS provides APIs for these operations:

- **GetRunningConfig:** This API is used to get the running configuration of the device.
- **GetStartupConfig:** This API is used to get the startup configuration. This method might not be valid for all the devices and for all the transport mechanisms.
- **UpdateRunningConfig:** This API is used to update the running configuration on a device.
- **WriteRunningToStartup:** This API is used to write the running configuration to the startup configuration.
- **OverwriteStartupConfig:** This API is used to overwrite the startup config of the device. This API is not valid for catalyst switches and for tftp.
- **ReloadDevice:** This API is used to reload the device. This API will be used to reload the device after changing the startup config.
- **ExecuteCommand:** This API use the telnet interface to execute given CLI commands on the remote device and returns the output received from the device as a String array. The commands are executed after entering the enable mode on the device.
- **CopyToStartupConfig:** This API over writes the startup configuration file of the remote device with the give configuration file. This API uses both telnet and the TFTP interfaces.
- **openTelnetLogin:** This API performs the telnet credentials check for successful entry into device telnet session at the login mode.

About the NT RCP Service

The RCP transport mechanism is the RCP client implementation for the transport operations to receive or update a device's configuration. You can use RCP to get both the startup and running configuration, but it supports IOS-based devices only.

The remote device needs to be configured with the local user, remote host, and remote user, using the following set of IOS commands:

```
configure terminal
ip rcmd rcp-enable
ip rcmd remote-host local-username {ip-address | host} remote-username enable
```

CISCO CONFIDENTIAL

RCP allows file copy between machines. RCP achieves this through remote command execution over RSH. RCP uses the local, remote username along with machine addresses to authenticate access to the remote machine; it does not need any additional password.

CWCS provides a basic RCP client. Using this service, you can copy files between local and remote machines. The service assumes that the remote machine is running an RCP server and proper permissions are set on the remote machine for remote access.

Table 25-3 describes the important RCP objects and methods.

Table 25-3 RCP Objects and Methods

Object	Description
Rcp Object	Rcp object provides methods to copy a file to and from the remote host. It creates a new RCP session for each copy operation and terminates the session once the copy operation is complete.
Rcp()	This is the Rcp Object constructor. It accepts remote host, rcp port, local username, remote username and rcp timeout. Throws an RcpException if any of values passed are null or timeout value is set to zero.
openConnection()	This a private method of Rcp Object that creates a RcpSocket connection to the remote host. This method also sets the timeout value of the connection.
closeConnection()	This a private method of Rcp Object that closes the existing Rcp connection to the remote host.
copyFromRemoteHost()	This method accepts remote and local filename. It opens a new Rcp connection to the remote host, copies the specified file from remote to local machine and closes the Rcp connection. Throws RcpException in case of any error.
copyToRemoteHost()	This method accepts remote and local filename. It opens a new Rcp connection to the remote host, copies the specified file from local to remote machine and closes the Rcp connection. Throws RcpException in case of any error.
RcpSocket	This a wrapper over a normal Socket class that provides buffering for input and output streams of the socket.
RcpSocket()	This constructor accepts a remote machine IP address, port number and timeout value. It opens a socket connection to remote machine on specified port and initializes the buffered streams of the connection.
close()	Closes the socket connection to the remote machine.
read()	Reads a byte or array of bytes from the socket.
readLine()	Reads a character line from the socket.
write()	Writes a string or array of bytes to the socket.
sendOK()	Sends a OK response to remote host.
expectOK()	Expects if the response from the remote host is <OK> (i.e. "0"). If it is not <OK> throws an exception with the message passed by remote host.
RcpException	RcpException will be used to signal any error condition occurring during the Rcp protocol operations.
RcpException()	Constructor of RcpException class. It accepts a message describing the details of exception.

CISCO CONFIDENTIAL

About the CRMLogger Service

CRMLogger is a common syslog collector that performs on Windows much the same services as syslogd performs on Solaris:

- It runs independently.
- It listens for syslogs from devices.

CRMLogger benefits applications by removing the syslog processing load and requiring devices to send only one copy of a syslog. While CRMLogger runs independently, it can run either remotely or locally on the machine where an application is running. For more details on the design of CRMLogger, see the *Common Syslog Collector Software Functional Specification*, EDCS-272409.

In this version of CWCS, CRMLogger has been written in C++/Java. It can be run as a Windows service or a command line tool. Installation of CRMLogger performs the registry key changes shown in [Table 25-4](#) automatically.

Table 25-4 CRMLogger Registry Keys

Key	Description
ClassPath	Sets the class path for CRMLogger's jakartha-oro and Java files.
CrmDnsResolution	Sets name resolution on or off. The default value is 0 (disabled). To enable DNS, set it to 1.
CrmLogPort	Sets the CRMLogger listening port. The default is 514.
CrmMaxHeapSize	Sets the maximum heap size for JVM. The default value is 256 MB.
CrmMinHeapSize	Sets the minimum heap size for JVM. The default value is 128 MB.
CrmMsgCount	Sets the number of messages to be read in one shot. The default is 256 messages.
DebugFile	Sets the name and path of the CRMLogger debug file. The default is NMSROOT\log\syslog_debug.log.
DebugLevel	Sets the CRMLogger debug message level. Settings of 2 and 4 are the only ones permitted; the default is 4. A setting of 2 means aggressive debugging, which will output all CRMLogger activities to the file specified in DebugFile. A setting of 4 means only warning messages are output.
FlushIntervalInMills	Sets the message flush interval (default value is 1 minute).
LogFile	Sets the name and path of the file that contains the processed syslog. The default value is NMSROOT\log\syslog.log.
MaxFlushCount	Sets the maximum processed messages allowed in the low-level buffer. The default is 100.
ProcessorThreadCount	Number of threads required for parsing messages according to RFC 3164. The default value is 5.
QueueCapacity	Sets the maximum size of the internal queue. This queue contains the unprocessed syslog messages from UDP socket. The default value is 100000. If the CrmMsgCount is 256, then the queue can contain up to 256 * 100000 messages.
SystemPath	Sets the path of the JRE.
UDPBufferSize	Sets the size of the UDP socket queue. The default is 15360 bytes.
UdpProcessorCount	Sets the number of threads required for reading messages from the UDP socket. The default value is 5.

The default values shown in [Table 25-4](#) are suitable for receiving 200 messages per second. In order to improve the performance up to 1,000 syslogs per second (at the network level), we need to tune the assigned values of the keys shown in [Table 25-5](#).

CISCO CONFIDENTIAL

To further optimize the I/O performance on the CRMLogger side, we have to consider tuning the values of FlushIntervalInMills and MaxFlushCount. The MaxFlushCount or FlushIntervalInMills decides when to write the processed syslog messages into the syslog.log file. Messages are written into the file when either of the parameters reached the limits specified in the registry.

Table 25-5 Tuning CRMLogger for 1,000 Syslogs/Second

Set this parameter	To this value
UDPProcessorThreads	7
RFCProcessorThreads	5
MaxFlushCount	2048
UDP Buffer Size	20 MB
CrmMsgCount	512

If you need to troubleshoot CRMLogger, try the following:

1. Go to the NMSROOT\bin folder and, using the console, stop the CRMLogger service. Then enter **crmlog** at the command prompt to launch CRMLogger from the command line. This is especially helpful if you suspect the trouble is due to problems with the Java environment.
2. Reset the DebugLevel registry key value to 2 and restart CRMLogger. This will allow you to see every CRMLogger operation.

Using CWCS NT Services

At installation, use the packaging information (.info) file to register for CWCS service bundles. CWCS services should be requested only by a *suite*, not a package.

Certain APIs are implemented by the installation team and can be used in package-specific hooks (install shell scripts for Windows).

Registering and Controlling NT Services

This is a set of functions and constants that allows to you to register and control NT Services.

Service Types

Use the following constants to specify Service Types:

- SERVICE_WIN32_OWN_PROCESS
- SERVICE_WIN32_SHARE_PROCESS
- SERVICE_WIN32
- SERVICE_INTERACTIVE_PROCESS

Service Start Types

Use the following constants to specify Service Start Type:

- SERVICE_BOOT_START
- SERVICE_SYSTEM_START

CISCO CONFIDENTIAL

- SERVICE_AUTO_START
- SERVICE_DEMAND_START
- SERVICE_DISABLED

Windows Services Functions

Use the following functions to register and control NT Services:

- [RegisterService](#), page 25-7
- [UnregisterService](#), page 25-7
- [StartService](#), page 25-8
- [ChangeServiceStartType](#), page 25-8
- [ChangeServiceAccount](#), page 25-8
- [ChangeService2Casuser](#), page 25-9
- [StopService](#), page 25-9

RegisterService

prototype **RegisterService** (STRING, STRING, STRING, LONG, LONG);
Registers new NT Services.

Arguments

Field	Type	Description
1	STRING (input)	Service name
2	STRING (input)	Service display name
3	STRING (input)	Path name of executable
4	LONG (input)	Service type. Choose from service type constants list (see Service Types , page 25-6).
5	LONG (input)	Service start type. Choose from service start type constants list (see Service Start Types , page 25-6).

UnregisterService

prototype **UnregisterService** (STRING);
Unregisters NT Services.

Arguments

Field	Type	Description
1	STRING (input)	Service name

CISCO CONFIDENTIAL**StartService**

prototype **StartService**(STRING);
Starts NT Services immediately.

Arguments

Field	Type	Description
1	STRING (input)	Service name

ChangeServiceStartType

prototype **ChangeServiceStartType**(STRING, LONG);
Changes service start type.

Arguments

Field	Type	Description
1	STRING (input)	Service name
2	LONG (input)	Service start type. Choose from service start type constants list (see Service Start Types , page 25-6).

ChangeServiceAccount

prototype **ChangeServiceAccount**(szServiceName, accountName,
accountPassword);

Changes the service account. Use this function after the services is created (see [RegisterService](#), page 25-7).

Arguments

Field	Description
szServiceName	The name of the service.
accountName	The account for the service to run as. To modify the service to run as LocalSystem account, the value should be “.LocalSystem” (without quotes, and remember to escape the backslash).
accountPassword	The password for the account. Leave this empty if the account is LocalSystem.

Related Topics

- [ChangeService2Casuser](#), page 25-9

CISCO CONFIDENTIAL**ChangeService2Casuser**

```
prototype ChangeService2Casuser (STRING, POINTER);
```

This function reconfigures a service to run as casuser. This relieves the developer from the need to fetch the value of the casuser password, which is maintained by the Daemon Manager. This function can only be used after the Daemon Manager has been installed.

Arguments

Field	Type	Description
1	String	The name of the service.
2	Pointer	The pointer to the string that contains the password. Specify NULL to use the password stored by the Daemon Manager.

Return Values

0 if successful.

Example

This function is implemented in secure.dll, which is a part of the installation framework. Therefore, a call to this function must be framed with the UseDll/UnUseDll calls. For example:

```
if (UseDLL(SUPPORTDIR ^ "secure.dll") != 0) then
    MessageBoxLog("Cannot load secure.dll", SEVERE);
endif;
ChangeService2Casuser("myService", NULL);
UnUseDll(SUPPORTDIR ^ "secure.dll");
```

StopService

```
prototype StopService (STRING, NUMBER);
```

Stops service.

Arguments

Field	Type	Description
1	STRING (input)	Service name
2	NUMBER (input)	Stop mode. Options: <ul style="list-style-type: none"> • RM_STOPSERV_IMMEDIATELY • RM_STOPSERV_DELAYED

Notes

- In preinstalls, use RM_STOPSERV_DELAYED. The installer does not stop services immediately; instead, it collects the list of services to stop.
- After all preinstalls are executed, the installer asks the user for confirmation and stops all requested services at once. In addition, the installer stops all services that were requested to be stopped.
- In all hooks other than preinstall, use the RM_STOPSERV_IMMEDIATELY mode.

CISCO CONFIDENTIAL**Writing Messages to Log Files**

There are two types of log files:

- The Process Manager log contains information regarding process start, termination, and Process Manager warning and error messages.
- The Application log stores information logged by an application. To write a message to the application log, direct the output to stderr/stdout.

The location of the log files is determined by the operating system:

- On Windows platforms:
 - The Process Manager log is located under NMSROOT/log/syslog.log.
 - Each application has its own application log under NMSROOT/log.
 - The application log has same name as the application with the extension.log.
- On Solaris platforms:
 - The Process Manager log is located at /var/adm/CSCOPx/log/dmgtd.log.
 - All applications share the same application log:
/var/adm/CSCOPx/log/daemons.log.



CISCO CONFIDENTIAL

CHAPTER 26

Using Device Center

CWCS Device Center provides an interface to invoke tools on a selected device from a single page. All information, tasks and reports for the device are made available at a single location. From the Device Center page, you can perform a variety of activities on a selected device, including:

- Change the device attributes
- Update device inventory
- Telnet to the device
- Launch Element Management tools, reports, and other management tasks.

The following topics explain Device Center and how to use it with your application:

- [Understanding Device Center](#)
- [Using Device Center With Your Application](#)

For basic information on Device Center, see the “[About Device Center Components](#)” section on [page 6-15](#).

For more information about Device Center, see:

- *CMF 2.3 PRD*, EDCS 263430
- *CMF 2.3 System Functional Specification*, EDCS-283137
- *CMIC Software Functional Specification*, EDCS 123728
- *Device Center Functional specification*, EDCS-285151
- *Device Center Usage Guidelines*, EDCS-323423
- *CMIC System Function Specifications*, EDCS-123728
- *CMIC API Definitions document*, EDCS-133481

Understanding Device Center

The information displayed on the Device Center comes from the CMIC registry. Applications register their management services with CMIC registry by defining a Management Service Template (MST), which has all the service URLs in a predefined XML format. The URLs, that need to be displayed on the Device Center, are tagged by keywords defined by the Device Center. The Device Center queries CMIC registry for these defined tags and the matched URLs will be shown in the Device Center.

CISCO CONFIDENTIAL

The information shown in Device Center is controlled by registrations with CMIC. This provides scalability in terms of features as and when installed (registered) with CMIC without any changes to the Device Center.

The following topics provide additional detail on:

- [What You Can Do With Device Center](#)
- [About Device Center Launch Points](#)
- [What's Inside Device Center](#)
- [About Device Center Dependencies](#)
- [About the Device Center Runtime Structure](#)
- [About the Device Center User Experience](#)

What You Can Do With Device Center

The Device Center allows you to:

- Provide and carry forward the features and functions that are generic troubleshooting tools based on what CWCS provides (for example, Ping and Traceroute)
- Operate on the devices managed by applications local to the server.
- Provide a device selector that allows the users to select a device from a list-tree as well as to select a device by entering its address or name.
- Provide a summary report on the selected device in addition to displaying available features and functions that can be launched against that device for troubleshooting or per-single device management purposes.
- Provide appropriate external APIs (through CMIC) for other products and applications to register and include additional Device Center features.

About Device Center Launch Points

You can launch Device Center using any of the following:

- User launches the Device Center main page from the CWHP and selects a device.
- Bookmark Device Center URL and launch directly from browser window.
- Device Center for a device from one of the application's functions (such as reports). For example, user launches Device Center by clicking the Device name from RME Inventory Reports.
- Third party applications by passing the device context as a parameter. Currently this is done from HPOV and Netview.

CiscoWorks users can launch the Device Center page either from the overview screen or from applications, directly calling the Device Center URL.

What's Inside Device Center

Device center contains the following modules:

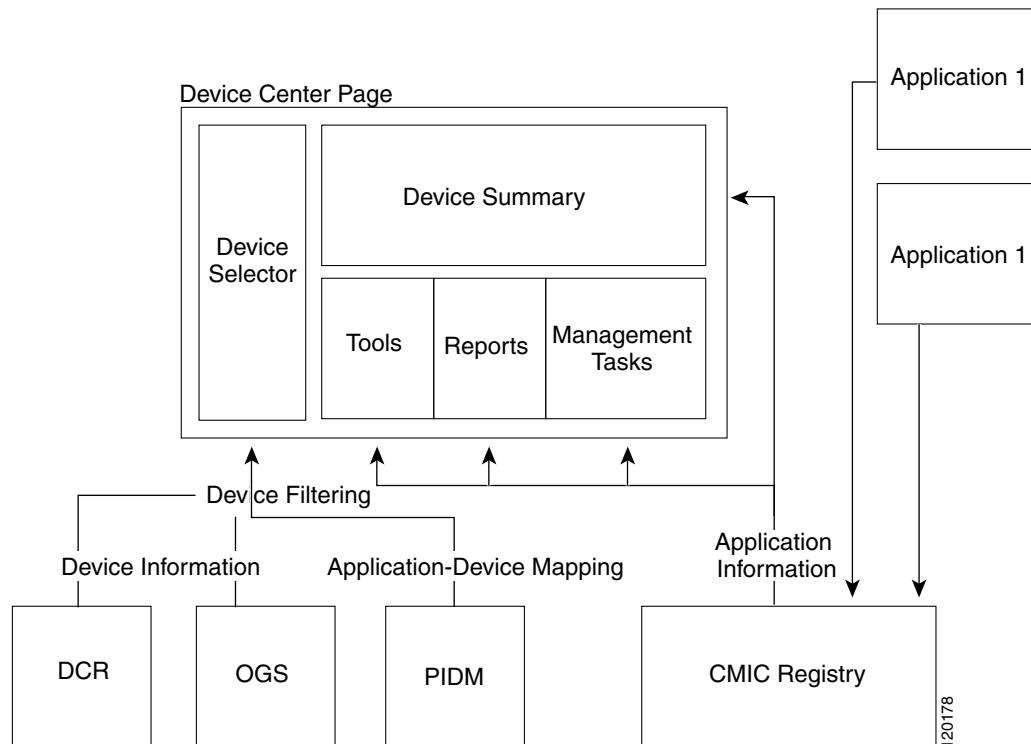
- Device Selector—The Device Selector module populates the devices for selecting them in Device Center.

CISCO CONFIDENTIAL

- **CMIC Interface**—The CMIC Interface module is used to query the CMIC registry to get the list of local URLs or links for those applications that has to appear in the main Device Center screen.
- **Summary Handler**—The Summary Handler module provides the device summary content in the Device Center. The device summary provides a basic snapshot summary about the device. For example, Last Reload Date, Last configuration change etc.
- **Reports Handler**—The Reports Handler module provides the list of the reports that can be launched for a selected device.
- **Tools Handler**—The Tools Handler module provides the list of debugging tools that can be used with the device. This module helps to debug device related problems.
- **Tasks Handler**—The Tasks Handler module provides the list of management tasks that can be performed on the Device.
- **Security Module**—The Security module helps to find whether the logged in user is authorized to perform the selected task.
- **UI Rendering Module**—The UI module gets the inputs from the Security module and displays the output using the browser in the Desired Format.

Figure 26-1 shows how Device Center concentrates application features on a single page.

Figure 26-1 Device Center Architecture



About Device Center Dependencies

Device Center depends on the following modules.

CISCO CONFIDENTIAL**Table 26-1** Device Center Dependencies

Module	Dependency
CMIC	To get the registered URL and APIs.
UII	User interface components.
Security Services	Authentication and authorization. Uses CAM.
Apache and Tomcat	Web server and servlet container.
Logging Service	Logging and debugging.
Device Credential Repository (DCR)	Fetching the list of devices and credentials.
PIDM	Information on Device-Application mapping.
OGS	Information on device grouping.

About the Device Center Runtime Structure

Device Center files are placed under:

\$NMSROOT/MDC/tomcat/webapps/cwhp/WEB-INF/classes/com/cisco/nm/cmfd/devicecenter

JSP files related to Device Center are placed under the following directory:

\$NMSROOT/MDC/tomcat/webapps/cwhp/screens/devicecenter

All Action classes are placed under the following directory:

\$NMSROOT/MDC/tomcat/webapps/cwhp/WEB-INF/classes/com/cisco/nm/cmfd/devicecenter/action

All Form Bean classes are placed under the following directory:

\$NMSROOT/MDC/tomcat/webapps/cwhp/WEB-INF/classes/com/cisco/nm/cmfd/devicecenter/actionform

All Utility classes are placed under the following directory:

\$NMSROOT/MDC/tomcat/webapps/cwhp/WEB-INF/classes/com/cisco/nm/cmfd/devicecenter/util

About the Device Center User Experience

User experience of Device Center varies depending on the following factors:

- **User launches Device Center link from CWHP**

A device selector and message that guides the user to the Device Center is shown in the content area. Device selector displays a list of devices managed by various applications that have registered links that can be shown in the Device Center on that server.

- **Only CWCS is installed on the local Server**

The Device Center will not display the device selector as there are no applications managing any device. The user can type IP address or host name in the field provided. It will display default tools that are provided with the Common Services.

- **Applications (RME, CM, and VHM) are installed on the local server and the user selects a device that is not managed by VHM**

Device Center shows links pertaining to RME and CM only. There will be no VHM links for the device.

CISCO CONFIDENTIAL

User selects a particular device that is managed by RME. A sub-task `Config-Archive` is shown which does not handle or support the device. In such cases, the respective task should display an appropriate error message.

- **Two applications have redundant information to show in summary page**

All the content that goes in to the Device Center will be finalized and approved by the Device Center team in consultation with the application teams. The summary information should not be duplicated.

- **For a selected device if there are no tools/tasks to show, but if there are summary information and reports**

The relevant sections are displayed. In this case the summary information is displayed with the Reports.

- **More than one application has summary information to show**

The summary information is grouped by applications. The user can click on an application to see the summary. When the user clicks on the summary, only the summary part of the page is refreshed.

Using Device Center With Your Application

The following topics describe how to integrate Device Center into your application:

- [Launching Device Center](#)
- [Registering Your Application With Device Center](#)
- [About Device Center Integration Tags](#)
- [Understanding PIDM](#)

Launching Device Center

The Device Center appends two parameters to the link that the application has specified for the task in the CMIC MST file. The parameters are:

```
deviceID = <String value of DCR Device ID [ DeviceId.getValue() ] >
deviceIP = <IP address or identity attribute of the Device [ such as hostname ]>
device = <Display name of the Device>
```

The applications should take these parameters and perform the required operations. For example, if the MST entry is

```
<TASKINFO TaskName="Chassis View" TaskIdentity="t002" TaskDescription="Device Management
" TaskCategory="O" TaskSubCategory="O/admin" SecurityTag="read" TaskURL="/CVng/chassis.do"
SubmitMet
hod="GET" IsAPI="true">
    <INTEGRATIONTAG TagName="DC_MANAGEMENT_TASKS">
    </INTEGRATIONTAG>
</TASKINFO>
```

The Device Center Link for this task is,

“/CVng/chassis.do?deviceID=12&deviceIP=10.77.210.22&device=test-pc”. This information is displayed in Device Center as a link with the name Chassis View under the sub-group Management Tasks.

CISCO CONFIDENTIAL

By default, all windows launched from the links in Device Center will not have Tool Bars and Menu Bars. If you want to display the pages of the application with Toolbars or Menu bars, the applications need to specify this in their application MST file as attributes for the each task:

```
<ATTRIBUTES Name="DC_TOOLBARS" Value="yes" />
<ATTRIBUTES Name="DC_MENUBARS" Value="yes" />
<ATTRIBUTES Name=" DC_LOCATIONBARS" Value="yes" />
```

For each link with these attributes, the Device Center window will be launched with a Tool bar and Menu bar.

Your application can launch Device Center by passing the URL in one of the following formats:

```
http://host:port/cwhp/device.center.do?DeviceID = <Device ID in String format
{DeviceId.getValue()}>
```

or

```
http://host:port/cwhp/device.center.do?device = <IP Address or display name of the System>
```

The Tools, Tasks, Reports & Summary for a particular Device can be launched in three ways:

- Select a device from the Device Selector.
- Enter a device IP address or device name in the text box provided, then click the **Go** button.
- Pass the device context as parameters for applications.

Registering Your Application With Device Center

Device Center does not have separate registration. It uses the CMIC registry to get the information on the applications.

To register an application with the Device Center:

Step 1 Create a valid MST Template.

CISCO CONFIDENTIAL

The MST Template is an XML file that lists all the tasks and URLs that the application intends to expose for integration with other components and applications. For more information on MST Template see the [“About the Device Center MST” section on page 26-7](#).

To use the MST Template for the Device Center, the template must contain Device Center Tags. For more information on Device Center Tags see the [“About Device Center Integration Tags” section on page 26-14](#).

The application should be a Cisco application in order to register with the Device Center. This can be set in the MST file with an `IsCisco="true"` tag.

- Step 2** Register the MST Template with CMIC. For details on this, see [Chapter 9, “Integrating Applications with CMIC”](#)
- Step 3** Provide mapping information about the devices managed by your application to the PIDM. For more information on PIDM, see the [“Understanding PIDM” section on page 26-17](#).
- Step 4** Provide necessary privileges to the links.
To provide privileges to the links, do a Task-Role mapping using `SecurityTag` attribute in the MST Template.

About the Device Center MST

The Management Service Template (MST) is organised under the tags, APPLICATIONRECORD, VENDORINFO, TASKGROUP, TASKINFO, and WSDL.

All attributes common to the applications are grouped under APPLICATIONRECORD.

Table 26-2 Attributes Under APPLICATIONRECORD

Attribute	Description
AppName	Application Name of the management service. For example, Campus Manager, Resource Manger Essentials.
AppVersion	Application Version of the management service. For example, 4.0, 3.5.
AppDescription	Brief description about the application
IsCiscoCertified	Specifies whether the application is Cisco certified. This field is reserved for future use. Leave it blank.
IsCisco	Specifies whether the application vendor is Cisco.
SecurityServiceIdentifier	The value of this field corresponds to how this application is identified when authorizing tasks of this application in Security context. Also known as “service Name” in the security tasks registration file. This is needed if some integration application wants to authorize URLs.
AppURL	The application level URL relative to the home page. Leave the field blank if the application does not have an AppURL.

CISCO CONFIDENTIAL**Table 26-2** Attributes Under APPLICATIONRECORD

Attribute	Description
Host, Port, And Protocol	These fields are left empty when the template is created; the API takes these as parameters while registering a service and fills them during the registration.
Modified Time, Modified User	These fields are left empty during template creation and are filled by CMIC during registration.

The vendor details of the applications are captured under the VENDORINFO tag. The table provides description of the attributes under VENDORINFO. The only mandatory attribute is VendorName.

The following table describes the attributes under VENDORINFO.

Table 26-3 Attributes Under VENDOR INFO

Attribute	Description
VendorName	Name of the application vendor.
Address	Contact address
Phone	Phone number
Fax	Fax number
E-mail	E-mail address
ContactURL	Contact URL of the vendor
SupportURL	Support URL of the vendor

All task attributes under an application are described in TASKINFO tag. An application can have multiple tasks associated with it. Each task is associated with one or more TASKGROUP in a nested manner. The task group indicates the functional hierarchy of the task starting from the first TASKGROUP tag. This allows multiple tasks to be grouped in a functional hierarchy.

**Note**

It is not necessary for a task to have a group hierarchy.

The following table describes the attributes under TASKINFO and TASKGROUP.

Table 26-4 Attributes under TASKINFO and TASKGROUP

Attribute	Description
TaskName	Name of the task
TaskCategory	This a broad category under which the task falls. The categories are Fault, Configuration, Accounting, Security, Performance, and Other. A task can be associated with one or more broad categories.

CISCO CONFIDENTIAL**Table 26-4** *Attributes under TASKINFO and TASKGROUP*

Attribute	Description
TaskSubCategory	The attribute sub category under which the task falls. A task can be associated with one or more sub categories.
SubmitMethod	The submit method for the task. The method can be of type Get or Post.
IsAPI	Indicates whether the task is a URL based API
TaskURL	URL of the task relative to web root.
SecurityTag	This tag is applicable to those services (CiscoWorks applications) that register their tasks with CiscoWorks security services. Security service allocates a unique identifier to recognize each individual tasks. The identifier allocated by security services should be a value of this attribute when the individual services create their MST. This attribute acts as a key between the CMIC registry and the security. This can be used to validate a set of tasks with security by passing security Tags.

Applications that need a closer integration with other applications should tag their tasks using the INTEGRATIONTAG.

For example, Device Center generates various reports on a device and finds out all such applications that can generate reports. It then defines an integration tag (say DC_REPORT). All the applications that generate reports will specify an element INTEGRATIONTAG with following attributes of their tasks.

A task can be associated with one or more integration tags. INTEGRATIONTAG element has the following attributes and sub-elements:

Table 26-5 *Attributes under INTEGRATION TAG*

Attributes	Description
Tag Name	The value of this field depends on the component you want this task to get integrated with. E.g. DC_REPORT, CWHP_TASK etc.

ATTRIBUTES is a sub-element of the integration tag. The attribute element has two attributes (Name and Value). This element is used to specify custom attributes for the integrating component (Device Center and Cisco Works Home Page). The tag notifies some specific parameters to the integrating component.

Each task has a Web Services Description Language (WSDL) tag. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.

CISCO CONFIDENTIAL

Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

The information captured using WSDL will decide the input for the services, the output, and the format of the input and output. WSDL has a comprehensive way of defining a web service.

Sample Device Center MST

Example 26-1 shows a sample MST suitable for use with Device Center.

Example 26-1 Device Center MST

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All information common to all tasks put here -->
<APPLICATIONRECORD AppName="Resource Manager Essentials" AppVersion="4.0"
AppDescription="An application remotely managing the devices in network "
IsCiscoCertified="true" IsCisco="true" Protocol="" Host="" Port="1" ModifiedTime=""
ModifiedUser="" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\Saras\CMIC\Docs\cmic_template.xsd">
<!-- Vendor related information -->
<VENDORINFO VendorName="Cisco Systems" Address="300 East Tasman Drive, San Jose
California" Phone="408 526-822" Fax="526 8222" Email="tac@cisco.com"
ContactURL="http://www.cisco.com" SupportURL="http://www-tac.cisco.com"/>
<!-- Task level information starts -->
  <TASKGROUP GroupName="RME">
    <TASKGROUP GroupName="SWIM">
      <TASKINFO TaskName="Add Image" TaskCategory="C" TaskSubCategory="C/image"
SecurityTag="rme.swim.addimage" IntegrationTag="CWHP_FUNC_TASK" TaskURL="/rme/addImage.do"
SubmitMethod="GET" IsAPI="false">
<!-- Type definitions -->
<types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="deviceContext">
      <xsd:sequence>
        <xsd:element name="deviceIP" type="xsd:string"/>
        <xsd:element name="WriteCommunity" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
<!-- Message definitions -->
<message name="AddImageRequest">
  <part name="DeviceContext" type="deviceContext"/>
</message>
<!-- Port type definitions -->
<portType name="AddImagePortType">
  <operation name="AddImage">
    <input message="AddImageRequest"/>
  </operation>
</portType>
<!-- Binding definitions -->
<binding name="AddImageHTTPGetBinding" type="AddImagePortType">
  <http:binding verb="GET"/>
  <operation name="AddImage">
    <http:operation location="AddImage"/>
    <input>
      <http:urlEncoded/>
    </input>
  </operation>
</binding>
</!--
```

CISCO CONFIDENTIAL

```

        <output>
            <mime:content type="text"/>
        </output>
    </operation>
</binding>
</definitions>
<xsd:element name="deviceIP" type="xsd:string"/>
    <xsd:element name="WriteCommunity" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
</types>
<!-- Message definitions -->
<message name="AddImageRequest">
    <part name="DeviceContext" type="deviceContext"/>
</message>
<!-- Port type definitions -->
<portType name="AddImagePortType">
    <operation name="AddImage">
        <input message="AddImageRequest"/>
    </operation>
</portType>
<!-- Binding definitions -->
<binding name="AddImageHTTPGetBinding" type="AddImagePortType">
    <http:binding verb="GET"/>
    <operation name="AddImage">
        <http:operation location="AddImage"/>
        <input>
            <http:urlEncoded/>
        </input>
        <output>
            <mime:content type="text"/>
        </output>
    </operation>
</binding>
</definitions>
</WSDL>
</TASKINFO>
</TASKGROUP>
</TASKGROUP>
</APPLICATIONRECORD>

```

MST XML-Schema

Example 26-2 shows a sample XML-Schema for a Device Center MST.

Example 26-2 Device Center MST XML Schema

```

<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<xs:element name="APPLICATIONRECORD">
    <xs:complexType >
        <xs:sequence>
            <xs:element ref="VENDORINFO"/></xs:element>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element ref="TASKGROUP" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element ref="TASKINFO" minOccurs="1" maxOccurs="unbounded"/>
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
</xs:element>

```

CISCO CONFIDENTIAL

```

    </xs:sequence>
<xs:attributeGroup ref="ApplicationRecordAttributes"> </xs:attributeGroup>
  </xs:complexType>
</xs:element>
<xs:element name="VENDORINFO">
  <xs:complexType>
    <xs:attributeGroup ref="VendorInfoAttributes"></xs:attributeGroup>
  </xs:complexType>
</xs:element>
<xs:element name="TASKGROUP">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="TASKGROUP" minOccurs="0" maxOccurs="unbounded"></xs:element>
      <xs:element ref="TASKINFO" minOccurs="0" maxOccurs="unbounded"></xs:element>
    </xs:sequence>
    <xs:attributeGroup ref="TaskGroupAttributes"></xs:attributeGroup>
  </xs:complexType>
</xs:element>
<xs:element name="TASKINFO">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="INTEGRATIONTAG" minOccurs="0" maxOccurs="unbounded"></xs:element>
      <xs:element ref="WSDL" minOccurs="0" maxOccurs="1"></xs:element>
    </xs:sequence>
    <xs:attributeGroup ref="TaskInfoAttributes"></xs:attributeGroup>
  </xs:complexType>
</xs:element>
<xs:element name="INTEGRATIONTAG">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ATTRIBUTES" minOccurs="0" maxOccurs="unbounded"></xs:element>
    </xs:sequence>
    <xs:attribute name="TagName" type="xs:string" default="required"></xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="WSDL">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace=http://schemas.xmlsoap.org/wsdl/ minOccurs="0"
maxOccurs="unbounded" processContents="lax"></xs:any>
    </xs:sequence>
    <xs:attribute name="Version" type="xs:string"></xs:attribute>
  </xs:complexType>
</xs:element>
<xs:element name="ATTRIBUTES">
  <xs:complexType>
    <xs:attribute name="Name" type="xs:string" use="required"></xs:attribute>
    <xs:attribute name="Value" type="xs:string" use="required"></xs:attribute>
  </xs:complexType>
</xs:element>
<xs:attributeGroup name="ApplicationRecordAttributes">
  <xs:attribute name="AppName" type="xs:string" use="required"/>
  <xs:attribute name="AppVersion" type="xs:string" use="required"/>
  <xs:attribute name="TemplateVersion" type="xs:string" use="required"/>
  <xs:attribute name="AppDescription" type="xs:string"/>
  <xs:attribute name="DisplayName" type="xs:string"/>
  <xs:attribute name="AppURL" type="xs:string"/>
  <xs:attribute name="AppURLWindowName" type="xs:string"/>
  <xs:attribute name="Host" type="xs:string"/>
  <xs:attribute name="Port" type="xs:integer"/>
  <xs:attribute name="Protocol" type="xs:string"/>
  <xs:attribute name="ModifiedUser" type="xs:string"/>
  <xs:attribute name="ModifiedTime" type="xs:string"/>
  <xs:attribute name="IsCisco" type="xs:boolean" default="true"/>

```

CISCO CONFIDENTIAL

```

<xs:attribute name="IsCiscoCertified" type="xs:string"/>
<xs:attribute name="SecurityServiceIdentifier" type="xs:string"/>
</xs:attributeGroup>
<xs:attributeGroup name="VendorInfoAttributes">
  <xs:attribute name="VendorName" type="xs:string" use="required"/>
  <xs:attribute name="Address" type="xs:string"/>
  <xs:attribute name="Phone" type="xs:string"/>
  <xs:attribute name="Fax" type="xs:string"/>
  <xs:attribute name="Email" type="xs:string"/>
  <xs:attribute name="ContactURL" type="xs:string"/>
  <xs:attribute name="SupportURL" type="xs:string"/>
</xs:attributeGroup>
<xs:attributeGroup name="TaskInfoAttributes">
  <xs:attribute name="TaskName" type="xs:string" use="required"/>
  <xs:attribute name="TaskIdentity" type="xs:string" use="required"/>
  <xs:attribute name="TaskCategory" type="xs:string"/>
  <xs:attribute name="TaskDescription" type="xs:string"/>
  <xs:attribute name="TaskSubCategory" type="xs:string"/>
  <xs:attribute name="SecurityTag" type="xs:string"/>
  <xs:attribute name="TaskURL" type="xs:string" use="required"/>
  <xs:attribute name="TaskURLWindowName" type="xs:string" />
  <xs:attribute name="SubmitMethod">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="GET"/>
        <xs:enumeration value="POST"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="IsAPI" type="xs:boolean" default="false"/>
</xs:attributeGroup>
<xs:attributeGroup name="TaskGroupAttributes">
  <xs:attribute name="GroupURL" type="xs:string"/>
  <xs:attribute name="GroupURLWindowName" type="xs:string"/>
  <xs:attribute name="GroupName" type="xs:string" use="required"/>
  <xs:attribute name="DisplayName" type="xs:string"/>
  <xs:attribute name="SecurityTag" type="xs:string"/>
</xs:attributeGroup>
</xs:schema>

```

Creating and Registering the MST With CMIC

To create and register an MST:

-
- Step 1** Create an MST.
The MST is an XML file which follows a specific [MST XML-Schema](#).
 - Step 2** Copy the MST to the <directory where you have installed CWCS>
<objects/data/cmf/cmhc/mst-templates.>
 - Step 3** Enter relevant information for the CMIC registration in Application registration page. You can access Application Registration from CWHF.
-

CISCO CONFIDENTIAL

About Device Center Integration Tags

For an application to show its links in Device Center, the application tasks must be tagged with Device Center integration tags. Table shows the integration tags for Summary, Tools, Reports and Management Tasks.

Table 26-6 *Device Center Integration Tags*

Integration Tag	Purpose
DC_DEVICE_SUMMARY	Links to the Device Summary section
DC_TOOLS	Links to the Tools section
DC_REPORTS	Links to the the Reports section
DC_MANAGEMENT_TASKS	Links to the Tasks section

The application tagging shown in [Example 26-3](#) will cause a “Monitoring Console” link to appear under the Tasks group in Device Center.

Example 26-3 *Integration Link*

```
<TASKINFO TaskName="Monitoring Console" TaskIdentity="t001"
TaskDescription="Console monitoring traps in network" TaskCategory="F"
TaskSubCategory="F/admin" SecurityTag="nm.dfm.monitor" TaskURL="/dfm/monitor.do"
SubmitMethod="GET" IsAPI="false">
  <INTEGRATIONTAG TagName=" DC_MANAGEMENT_TASKS ">
  </INTEGRATIONTAG>
</TASKINFO>
```

Application teams must create the template with appropriate values. The template must be sent to mst-police@cisco.com. This alias comprises all the component integration owners. They will assist the application team in creating a valid template and checking for syntactical errors and they would provide suggestions to improve the template. The review-team will not validate the content. They will only validate syntax and check if appropriate tags are provided for integration with their respective components.

Currently the identified integration component owners are Cisco Works Home page team (cwhp-dev), and the Device center team (devcenter-dev).

About UII Rendering Module

**Note**

The UII module uses UII version 6.0 for the final output.

The following table summarizes the various Device Center Integration tags:

CISCO CONFIDENTIAL

To bypass the PIDM checks, the application must import the class `com.cisco.nm.pidm.DeviceCenterPIDMRegister`, located under the `/lib/classpath` directory of the CWCS installation. Table 26-8 shows the APIs. In all these APIs, “Appname” is the name of the application as given in the MST file.

Table 26-8 APIs Used for PIDM Bypassing

API	Description
<code>int removePIDMCheckForDC(String Appname)</code>	Bypasses PIDM checking in DC for application Appname. Returns <i>0</i> if the operation is successful, <i>-1</i> if it fails.
<code>int addPIDMCheckForDC(String Appname)</code>	Specifies that Appname is doing PIDM registration. Required only for application that have previously removed the PIDM check and now want to restore it. Returns <i>0</i> if the operation is successful, <i>-1</i> if it fails.
<code>boolean isAppDoingPIDMRegistration(String Appname)</code>	Checks if an application is registering with PIDM. Returns <i>true</i> if the application is doing PIDM Registration, <i>false</i> if application is bypassing PIDM registration.



CISCO CONFIDENTIAL

CHAPTER 27

Using Product Instance Device Mapping

Product Instance Device Mapping (PIDM) is a registry that stores mapping information between devices and applications.

Applications need to register with PIDM the information about the devices managed by the applications. Applications can do this using PIDM APIs (which are also available via CSTM from Common Services 3.0 Service Pack 2).

Device Center depends on this information for showing the devices in Device Selector and also for showing the information under Tools , Management Tasks , Reports and Summary sections.

Device Center usually shows only the devices managed by the applications in the Device Selector. On selecting a device, the links corresponding to the applications that manage the device are shown in Device Center.

The following topics describe how to integrate DCR with your application:

- [Using the PIDM APIs](#)
- [Using the PIDM North-bound APIs](#)

Using the PIDM APIs

The PIDM APIs are as follows:

Table 27-1 PIDM APIs

APIs	Attributes
<code>com.cisco.nm.dcr.AppId[]</code> <code>getMappedAppIDs(com.cisco.nm.dcr.DeviceId deviceID)</code>	Returns the managing application's IDs for the given device.
<code>com.cisco.nm.dcr.AppId[]</code> <code>getMappedAppIDs(com.cisco.nm.dcr.DeviceId deviceID)</code>	Returns the deviceIDs for the devices managed by the given application.
<code>void mapDevicesToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId[] deviceIDs)</code>	Marks given devices as 'managed by' given application.
<code>void mapDeviceToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId deviceID)</code>	Marks given device as 'managed by' given application.

CISCO CONFIDENTIAL**Table 27-1** PIDM APIs

APIs	Attributes
void unmapDevicesToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId[] deviceIDs)	Removes the mapping between given application and devices.
void unmapDeviceToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId deviceID)	Removes the mapping between given application and device.
void close()	Closes the pidm proxy.

Creating the ProductToDeviceMapProxy Object

```
ProductToDeviceMapProxy pidm = new ProductToDeviceMapProxy();
```

Mapping a Device or Marking a Device(s) as Managed

```
DeviceId devId = new DeviceId("101");
AppId Myapp = new AppId("Campus Manager", "4.0", "bundle-pc10");
try {
    pidm.mapDeviceToProduct(Myapp, devId);
} catch (PDMEException pdmExp) {
    System.out.println("Error in mapping a Device " + pdmExp.getMessage());
}
```

Unmapping a Device or Marking a Device(s) as Not Managed

```
DeviceId devId = new DeviceId("101");
AppId Myapp = new AppId("Campus Manager", "4.0", "bundle-pc10");
try {
    pidm.unmapDeviceToProduct(Myapp, devId);
} catch (PDMEException pdmExp) {
    System.out.println("Error in un-mapping a Device " + pdmExp.getMessage());
}
```

Retrieving PIDM Information

To get the applications managing a particular device:

```
DeviceId devId = new DeviceId("101");
AppId appIds[] = pdm.getMappedAppIDs(devId);
// To get devices managed by a particular application.
DeviceId devIds[] = pdm.getMappedDeviceIDs(Myapp);
```

CISCO CONFIDENTIAL

Using the PIDM North-bound APIs

PIDM APIs are also available via CSTM from Common Services 3.0 Service Pack 2.

PIDM North-bound APIs

The PIDM North-bound APIs are as follows:

Table 27-2 PIDM North-bound APIs

APIs	Attributes
com.cisco.nm.dcr.AppId[] getMappedAppIDs(com.cisco.nm.dcr.DeviceId deviceID, com.cisco.nm.dcr.APIExtraInfo apiExtraInfo)	Returns the managing application IDs for the given device.
com.cisco.nm.dcr.DeviceId[] getMappedDeviceIDs(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.APIExtraInfo apiExtraInfo)	Returns the device IDs for the devices managed by the given application.
void mapDevicesToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId[] deviceIDs, com.cisco.nm.dcr.APIExtraInfo apiExtraInfo)	Marks given devices as managed by given application.
void mapDeviceToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId deviceID, com.cisco.nm.dcr.APIExtraInfo apiExtraInfo)	Marks given device as managed by given application.
void unmapDevicesToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId[] deviceIDs, com.cisco.nm.dcr.APIExtraInfo apiExtraInfo)	Removes the mapping between given application and devices.
void unmapDeviceToProduct(com.cisco.nm.dcr.AppId appID, com.cisco.nm.dcr.DeviceId deviceID, com.cisco.nm.dcr.APIExtraInfo apiExtraInfo)	Removes the mapping between given application and device.
void close()	Closes the PIDM proxy.

PIDM NB APIs and Associated Tasks

The PIDM NB APIs and associated tasks are as follows:

Table 27-3 PIDM NB APIs and Associated Tasks

API	DCR Task	ACS Task
getMappedDeviceIDs	VIEW_DEVICE_TASK	View
getMappedAppIDs	VIEW_DEVICE_TASK	View
mapDeviceToProduct	ADD_DEVICE_TASK	Add
mapDevicesToProduct	ADD_DEVICE_TASK	Add

CISCO CONFIDENTIAL**Table 27-3** *PIDM NB APIs and Associated Tasks*

API	DCR Task	ACS Task
unmapDeviceToProduct	DELETE_DEVICE_TASK	Delete
unmapDevicesToProduct	DELETE_DEVICE_TASK	Delete

Creating the APIExtraInfo Object

ProductToDeviceMapNBProxy API accepts objects of class APIExtraInfo in addition to their regular arguments. APIExtraInfo encapsulates:

- The AppID - This object is the unique ID of your application, and establishes your application name, version number, and host information for use in the SourceContext object.
- The SourceContext - This object establishes your application and its context as the source of the ProductToDeviceMapNBProxy API call.
- The SecurityContext - This object contains the information needed to authenticate or authorize the ProductToDeviceMapNBProxy API request.

```
AppId Myapp = new AppId("Device Manager", //unique AppID and application name
    "1.3.2", //application version number
    "192.168.1.15"); // host on which the application is running

SourceContext source = new SourceContext(Myapp) ;
// For Local API calls
SecurityContext security = new SecurityContext("username");

// For North-Bound API calls
SecurityContext security = new SecurityContext("username",
    "password",
    "secretKey")
//Wrapper for Source and Security information:
APIExtraInfo extraInfo = new APIExtraInfo(security, source);
```

Creating the ProductToDeviceMapNBProxy Object

```
ProductToDeviceMapNBProxy pidmNB = new ProductToDeviceMapNBProxy();
```

Mapping a Device or Marking a Device(s) as Managed

```
DeviceId devId = new DeviceId("101");
AppId Myapp = new AppId("Campus Manager", "4.0", "bundle-pc10");

try {
    pidmNB.mapDeviceToProduct(Myapp, devId, extraInfo);
}
catch (PDMEException pdmExp) {
    System.out.println("Error in mapping a Device " + pdmExp.getMessage());
}
```


CISCO CONFIDENTIAL

Unmapping a Device or Marking a Device(s) as Not Managed

```
DeviceId devId = new DeviceId("101");
AppId Myapp = new AppId("Campus Manager", "4.0", "bundle-pc10");

try {
    pidmNB.unmapDeviceToProduct(Myapp, devId, extraInfo);
}
catch (PDMEException pdmExp) {
    System.out.println("Error in un-mapping a Device " + devId + " : " + pdmExp.getMessage());
}
```

Retrieving PIDM Information

To get the applications managing a particular device:

```
DeviceId devId = new DeviceId("101");
AppId appIds[] = pidmNB.getMappedAppIDs(devId, extraInfo);
```

To get devices managed by a particular application.

```
DeviceId devIds[] = pidmNB.getMappedDeviceIDs(Myapp, extraInfo);
```

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

CHAPTER 28

Integrating Applications With Device Selector

The Device Selector is used to select devices to perform various device management tasks. This lists all devices in a group. The Display Name of the devices entered when you have added the devices in DCR is displayed as the device name in the Device Selector.

The Device Selector is enhanced in Common Services 3.0.5 to address the usability issues. The enhancements to Device Selector include the following:

- Search and Advanced Search functionality
- Group Customization and Group Ordering Feature
- Tooltips for device names
- Hyperlink to the message “x devices selected”

Refer to the User Guide for CiscoWorks Common Services 3.0.5 for more information on new features and enhancements.

DCR and Device Center is integrated with the enhanced Device Selector for performing the device management functions in Common Services 3.0.5.

Individual applications should integrate with the enhanced Device Selector delivered as part of Common Services 3.0.5 and User Interface Infrastructure kit 6.3 for their device management tasks.

This chapter explains you the following details of integration with New Device Selector:

- [UII Integration](#)
- [Integration with Search feature](#)
- [Integration with Advanced Search Feature](#)
- [Integration with Tree Generator](#)



Note

Javadocs provides the details of all APIs and the complete set of customizations that can be made to Search and Advanced Search implementation. Applications can override few of the APIs to suit to their requirements.

CISCO CONFIDENTIAL

UII Integration

UII tag libraries are enhanced and new tags are added to the existing HTML Object Selector (HOS) to add the support for Search and Advanced search.

The following new properties are introduced in the `uii:hosContentAreaSelector` component.

- `showSearch` - The value must be set to true to enable the search feature.
- `showAdvancedSearch` - The value must be set to false to enable the advanced search feature
- `searchHandler` - This property refers to the implementation class that provides the implementation of search. In other words, this refers to the implementation class of `com.cisco.nm.uii.hos.SearchHandler` interface

UII provides the enhanced APIs for applications to set the Search results as a result of search or advanced search operation.

Applications must add the following code in their classes apart from the set of UII calls to enable the search behaviour:

```
else if (HOSRequestHelper.isRequestForSearch(request))
{
    return HOSRequestHelper.doSearch(request,mapping);
}
else if (HOSRequestHelper.isRequestForSaveSelections(request)) {
    return HOSRequestHelper.saveSelectedEntriesAndForward(request ,null ,mapping);
}
```

Refer to UII 6.3 SDK documentation for more information on UII Integration with Search and Advanced Search feature.

Integration with Search feature

You can enter your search criteria in the Search Input field of Device Selector and search for the devices using the Search icon. The search results are based on the display name of the devices added in DCR.

Common Services provides the default implementation for the `SearchHandler` interface of UII.

Applications should extend this implementation class `com.cisco.nm.cmf.devsel.SearchHandlerImpl` to validate the search criteria specified, search the CMF database based on search criteria, get the list of devices managed by the local application and to filter the results based on the application contexts.

Applications should extend the implementation class to provide the following features:

- Perform device based authorization - You should set the `OGSSecurityContext` in order to mention the `TaskId` and `ApplicationId` values for performing device based authorization.
- Provide any custom search - If an application wants to search on another database in addition to DCR, it should extend the implementation class provided by CMF or implement the UII `SearchHandler` interface directly.

Refer to the Javadocs for the API specifications for all the classes provided by Common Services.

Application can extend the Common Services classes and customize their implementation as required.

CISCO CONFIDENTIAL

Configuring Property files

In addition to extending the Common Services classes, Applications need to configure few properties in the following properties files:

1. DeviceSelector.properties - You should specify a set of properties in this file for the search feature to function.
 - a. OgsUrn - This property defines the URN of the OGS Server. Enter the value only when your application have a URN different from Common Services.
 - b. OgsServerName - This property defines the name of the local OGS Server used for searching devices. The OGS Server Name used for Common Services is CMFOGSServer.
 - c. OgsRelativeURL - Defines the relative URL of the CTMServlet for the local OGS Server. For example, you can enter the property value as /cwhp/CTMServlet.
 - d. AllDevicesGroup - Defines the name of the All Devices group that is used for filtering based on applications or tasks after a device search.
 - e. SetMDFIcon - Defines whether the icons needs to set for the Search results. Set the value as Yes, if you want to display the search results (device names) with their icons. Default value is No.
2. log4j-devsel.properties - Properties must be specified to define the log levels of the Common Services search or advanced search implementations.
 - a. -log4j-devsel.properties: For defining the log levels of the CS search / advanced search implementations.
 - b. log4j.category.com.cisco.nm.cmf.devsel=DEBUG, DevSel
 - c. log4j.category.com.cisco.nm.cmf.devsel.util=DEBUG, DevSel
 - d. log4j.additivity.com.cisco.nm.cmf.devsel=false
 - e. log4j.appender.DevSel=org.apache.log4j.RollingFileAppender
 - f. log4j.appender.DevSel.File=CSDeviceSelector.log
 - g. log4j.appender.DevSel.layout=org.apache.log4j.PatternLayout
 - h. log4j.appender.DevSel.layout.ConversionPattern=%d{dd/MMM/yyyy HH:mm:ss:SSS} %-4r [%t] %-5p %c %x %L - %m%n

Integration with Advanced Search Feature

You can use the Advanced Search icon to open the Advanced Search popup window and specify a set of rules for performing an Advanced search. The advanced search is based on the grouping attributes of the application's grouping server. For example, when you launch an Advanced Search from Campus Manager Device Selector, the attributes of the Campus Manager Grouping server appears.

Common Services provides the default implementation of View, Form and Action classes that connects to the respective local OGS Servers and provide the default rule based search.

Applications need to configure the following to integrate the Advanced Search feature:

1. Applications need to configure few property files to let the Common Services Implementation know about the details of the OGS Server. Refer to [Configuring Property files](#) for the list of property files and properties to be configured.
2. Applications should deliver the JSP files DeviceFilter.jsp and closeDeviceFilter.jsp.

CISCO CONFIDENTIAL

3. Application should define the struts-config and sitemap XML files to configure the mapping between the form bean and action classes, and to set other parameters.

- a. Changes to struts-config.xml

Form bean entries:

```
<!-- ##### DeviceSelector Specific ##### -->
<form-bean name="DeviceFilterForm" type="com.cisco.nm.cmf.devsel.DeviceFilterForm"
/>
<!-- ##### DeviceSelector Specific END ##### -->
```

Global forward entries:

```
<!-- ##### DeviceSelector Specific ##### -->
<forward name="DeviceFilter" path="/WEB-INF/screens/popup.jsp?sid=DeviceFilter" />
<forward name="closeDeviceFilter"
path="/WEB-INF/screens/deviceselector/closeDeviceFilter.jsp" />
<!-- ##### DeviceSelector Specific END ##### -->
```

Action mapping entries:

```
<!-- ##### DeviceSelector Specific ##### -->
<action path="/DeviceFilter" scope="request" name="DeviceFilterForm"
validate="false" type="com.cisco.nm.cmf.devsel.DeviceFilterAction">
</action>
<!-- ##### DeviceSelector Specific END ##### -->
<action path="/hosTreeContent" name="nullFB"
type="com.cisco.nm.uui.hos.action.HOSContentAreaAction"/>
```

- b. Changes to sitemap.xml

Add the following entries in the application pages containing the HOS component with the advanced search:

```
<appContentArea screenID="DeviceFilter" contentAreaTitle="Advanced Search"
helpTag="" fileRef="/WEB-INF/screens/deviceselector/DeviceFilter.jsp" />
```

4. Applications should define the Javascript function in their JSP files to launch the advanced search popup screen when you click the advanced search icon.

The Javascript function is as follows:

```
function advancedSearchBtnOnClickHandler(osName) {
    window.open("/cwhp/DeviceFilter.do?treeID="+osName, "newer", width=580,
height=350, scrollbars=no,resizable=yes",true); }
function submitForm() {
    window.frames['contentFrame'].document.forms[0].submit(); }
```

Application can go for any custom implementation and create new action classes. To accomplish that applications should call an API in HOS to provide the search results. This API in HOS needs to be called only if the search operation is successful. Refer to UII 6.3 SDK documentation for more information.

CISCO CONFIDENTIAL

Integration with Tree Generator

The Tree Generator is enhanced to provide a new default tree view to end users with new device groups such as Application Specific Groups and User Defined Groups in the tree. The Tree Generator also provides a customized tree view as per the configurations saved by end users while using the Group Customization and Group Ordering feature.

Common Services provides the Tree Generator implementation for the other applications to integrate.

This section explains the following:

- [Tree Generator Changes for Device Selector Nodes](#)
- [Tree Generator Changes for Search Implementations](#)
- [Tree Generator Changes for Group Customization and Group Ordering](#)

Tree Generator Changes for Device Selector Nodes

Since Common Services does not have the “All Devices” Node or Group, the existing Tree Generator is extended to add an “All Devices” group.

Tree Generator now extends `com.cisco.nm.xml.ogs.client.ostaglib.util.AbstractTreeStateManager` class, which adds a new API to return the id of a group containing the devices of all application. Applications that have the device group “All Devices” can return the same group for this new API or can extend the existing Tree Generator classes.

Apart from this, Applications should extend the Tree Generator classes to add the application specific groups to the Device Selector or when they need a tree view specific to the application.

Tree Generator Changes for Search Implementations

Tree Generator sets the `OGSSecurityContext` object in the `HttpSession` as required by the Search and Advanced Search implementations and is used in all API calls to Object Grouping Services Server.

Tree Generator Changes for Group Customization and Group Ordering

Based on the Group Customization and Group Ordering settings entered by an end user, the Device Selector should load a customized tree view to the logged in user.

Common Services provides a base implementation for the new tree generator which uses the Group Customization and Group Ordering feature. Applications should use the Tree Generator class `com.cisco.nm.cmf.devsel.DeviceSelectorTreeGenerator` provided by Common Services.

Few classes are added to the Device Selector util and preference packages for the new Tree Generator. The Table below lists the classes added to Device Selector packages and their description.

CISCO CONFIDENTIAL**Table 28-1** *New Classes added to Device Selector packages for Tree Generator*

Package	Class	Description
com.cisco.nm.cmf.devsel.util	Group	Stores the mapping between group id and group name
	GroupCollection	Collection class for the group object
	GroupMembership	Stores the membership of group and this is an extension class of Group
	GroupMembershipCollection	Collection class for the Group Membership object
com.cisco.nm.cmf.devsel.preference	DeviceSelectorPreferences	Stores the device selector settings entered by an end user.
	DeviceSelector	Stores and retrieves the device selector preferences entered. This is a Main class exposing the APIs.
	DeviceSelectorConstants	Defines the constants to be used in DeviceSelector class. For example, ALL_DEVICES.

Application needs to integrate this implementation to communicate with the respective local OGS Server, get the user preferences from the database, return the top-level groups and return the devices for the top-level groups.



CISCO CONFIDENTIAL



PART 1

About CWCS Per-Product Services



CISCO CONFIDENTIAL

CHAPTER 29

Using Per-Product Services

Per-Product Services are additional, value-added services that individual products can choose to include based on their particular requirements. Installations of Per-Product Services are normally not shared among the applications installed on a single server.

The following topics discuss the CWCS Per-Product Services and the components you need to include to use them as part of your applications:

- [Understanding Per-Product Services](#)
- [About the Per-Product Services Components](#)

Understanding Per-Product Services

The Per-Product Services discussed in this section are always installed and run by individual applications. They form a set of advanced services that add value to applications by either meeting requirements for advanced functions in CiscoWorks applications, or allowing for variation in how these features are implemented. [Table 29-1](#) summarizes these feature requirements and the corresponding Per-Product Services designed to meet them.

Note that some Per-Product components (such as UII) are also used in the CWCS-R distribution. This is done to support certain Shared Services. Note that these CWCS-R-distributed Per-Product components are for CWCS use only. Applications may *not* share Per-Product components installed with CWCS-R. Application teams must install their own copies of Per-Product components with their applications, and they may not modify the source of or substitute products intended to perform the same functions as any Per-Product component distributed with CWCS-R.

As noted in the table, many Per-Product components are provided as part of the CWCS-SRC distribution. Application teams may modify CWCS-SRC-distributed Per-Product components for their own requirements, and install them with their applications. Note that it is a requirement of CWCS-SRC distribution that application teams who modify a CWCS-SRC component's source are responsible for providing their own support on these components. Application teams must also provide fully commented copies of their modified source to the CWCS Team.

Per-Product Services often represent the future direction of Common Services. Where possible, you should plan your application to adopt them as standard. For example: the CiscoWorks Home Page, which provides GUI access to applications, has been adopted as the standard GUI, entirely replacing the old CMF Desktop.

CISCO CONFIDENTIAL**Table 29-1 Per-Product Services: Feature Requirements and Services Map**

Category	Requirement	Description	Type	Per-Product Service
Graphic User Interface	User Interface Support	Support a web-based launch point that aggregates all applications.	CWCS-SRC	The launch point is provided by the CiscoWorks Home Page (see the “About the CiscoWorks Home Page Component” section on page 6-5), which in turn requires the User Interface Infrastructure (UII). See the “About the User Interface Infrastructure” section on page 29-7.
	Device Selector	Let users see and select devices and groups of objects	CWCS-SRC	Part of the UII; comes in several different forms. See the “About the User Interface Infrastructure” section on page 29-7.
	Graphing Service	Allow GUI display of graphs from user data.	CWCS-SRC	A Graphing Framework is provided as part of the UII. See the “About the User Interface Infrastructure” section on page 29-7.
	Group Management	Let users or administrators create and maintain groups of objects (e.g., devices or user IDs).	CWCS-SRC	See the “About the Object Grouping Service (OGS) Components” section on page 29-4.
Operations	Device Package Update	Support delivery of incremental device packages.	CWCS-R	Provided by PSU with VDS. See the “About the Package Support Updater (PSU) Components” section on page 29-5
	Incremental Device Support	Permit incremental device support.	CWCS-SRC	Provided by the CIDS package. See the “About the Common Incremental Device Support (CIDS) Component” section on page 29-5
	IPC/RPC Support	Provide support for Inter-Process Communications (IPC), In-Process Calls, and Remote Procedure Calls (RPC)	CWCS-SRC	CSTM provides full IPC/RPC support. See the “About the Common Services Transport Mechanism (CSTM) Components” section on page 29-4.
	Licensing	Provide flexible management of licenses, including multiple keys.	CWCS-R	See the “About CWCS Licensing” section on page 29-6.
	SOAP Support	Support SOAP encoding.	CWCS-SRC	CSTM uses SOAP internally and exposes northbound APIs in a SOAP-based format. See “About the Common Services Transport Mechanism (CSTM) Components” section on page 29-4
	Virtual Download Service	Download and install incremental device packages.	CWCS-R	Provided by PSU with VDS. See the “About the Package Support Updater (PSU) Components” section on page 29-5

CISCO CONFIDENTIAL**Table 29-1 Per-Product Services: Feature Requirements and Services Map (continued)**

Category	Requirement	Description	Type	Per-Product Service
Utilities	DOM Parser	Provide Document Object Model parser support for Core, MC and Kilner applications	CWCS-R	JDOM 1.0 Beta 8 is supported. See the "" section on page 29-8 .
	Email Support	Provide utilities that support email generation	CWCS-R	Utilities provided include JavaMail 1.2 and Blat for Windows NT. Solaris email is used on Solaris platforms. See the "" section on page 29-8 .
	JRE Support	Support multiple versions of the Java Runtime Environment.	CWCS-R	JRE versions 1.2.2, 1.3.1, and 1.4 are supported. See the "" section on page 29-8 .
	Logging Service	Provide utilities that support creation and management of application logs.	CWCS-R	Utilities provided include Log4J 1.1.3, coreLog4c, CoreLogger, CMF logging, and JGL 3.1. See the "" section on page 29-8 .
	SNMP Library	Support SNMP calls from Java applications.	CWCS-SRC	The latest version of the SNMPOnJava library supports SNMPv3. See the "" section on page 29-8 .
	XML Parser	Support a variety of XML parsers.	CWCS-R	Libraries provided include standard Xerces Java 1.4.3 and 1.4.4 library, JAXP, Crimson, Xerces C++ 1.5.1, and IBM4J. See the "" section on page 29-8 .
	XSLT Support	Support a variety of XML Stylesheet processors.	CWCS-R	Libraries provided include LotusXSL 0.16.3, Xalan Java 2.2.d9, Xalan C++ 1.5.1, and Xalan 2.4.1. See the "" section on page 29-8 .

About the Per-Product Services Components

The following topics provide basic information for each of the Per-Product Service components:

- [About the Object Grouping Service \(OGS\) Components](#)
- [About the Common Services Transport Mechanism \(CSTM\) Components](#)
- [About the Package Support Updater \(PSU\) Components](#)
- [About the Common Incremental Device Support \(CIDS\) Component](#)
- [About CWCS Licensing](#)
- [About the User Interface Infrastructure](#)

Each topic includes:

- Basic information about the component's purpose and features.
- Usage guidelines. Where these guidelines require a separate chapter in this *Guide*, a pointer to that chapter is supplied.
- Pointers to Java, C++, Perl or other references and code samples supplied as part of the SDK.
- The names of the packages on which the component depends.
- Where they exist: The names of utilities that help you troubleshoot or use the component.

For the same information for Shared Services, see [Chapter 6, "Using Shared Services"](#).

CISCO CONFIDENTIAL**About the Object Grouping Service (OGS) Components**

The Object Grouping Service provides a generic framework for creating, managing, and sharing hierarchical groups of objects. These objects can be any type of data object, from network devices to user IDs, capable of being grouped based on shared attributes. The OGS framework provides tools to define groups useful to your application, and to supply them in predefined form with your application.

For guidelines to follow when including OGS with your application, see [Chapter 30, “Using Object Grouping Services”](#). OGS is functionally dependent on the packages shown in [Table 29-2](#).

Table 29-2 OGS Package Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOdab/db	Common Services Database (includes diskwatcher, DB wrappers)
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOgrid/grid	Grid
CSCOjaws/jaws	JSCAPE JavaAWT for widgetd
CSCOjcht/jchart	Jchart Java Class
CSCOjpwr/jpwr	JSCAPE Power Search Classes
CSCOjre2/jre2	CMF JRE 1.2.2
CSCOl4j/log4j	Log4j Logging Framework
CSCOmddmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOswing2/swing2	Java Swing2 Package
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOxsl/lotusxsl	Lotus XSL Engine Classes
CTM	Common Services Transport Mechanism
GS	Graphing Service
OGS	Object Grouping Service
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail for NT)
UII	User Interface Infrastructure

About the Common Services Transport Mechanism (CSTM) Components

CSTM (formerly Common Transport Mechanism, or CTM) is a simple, platform-agnostic method for performing all types of inter-program communications. It does not require that sender or receiver applications implement proprietary standards like CORBA or DCOM, use the same object model, have full knowledge of message contexts, or avoid encoding metadata. Instead, it abstracts these

CISCO CONFIDENTIAL

communications, provides a single API for dealing with all of them, and bases the API on well-known non-proprietary communications and data standards, including XML, Serialized Java Objects, HTTP, sockets, and standard binary and SOAP encoding.

For guidelines to follow when including CSTM with your application, see [Chapter 31, “Using the Common Services Transport Mechanism”](#). CSTM is functionally dependent on the packages shown in [Table 29-3](#),

Table 29-3 *CSTM Package Dependencies*

Package Name	Description
CSCOjre2/jre2	CMF JRE 1.2.2
CSCOjrun/jrun	JRun Servlet Engine
CSCOlg4j/log4j	Log4j Logging Framework
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
CTM	Common Services Transport Mechanism

About the Package Support Updater (PSU) Components

The PSU (and associated Virtual Download Service, or VDS) allows your application to:

- Check for software and device support updates.
- Download these updates and related dependent packages to the application’s server file system.
- Install them.

For guidelines to follow when including PSU/VDS with your application, see [Chapter 32, “Using Package Support Updater”](#). PSU is functionally dependent on the packages shown in [Table 29-4](#).

Table 29-4 *PSU Package Dependencies*

Package Name	Description
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
PSU/VDS	Package Support Updater/Virtual Download Service

About the Common Incremental Device Support (CIDS) Component

Common Incremental Device Support (CIDS) provides a method of adding device information to any network management application without requiring re-installation of the entire product. CIDS provides an application layer that encapsulates all device information and update mechanisms, permitting applications that include a CIDS layer to be incrementally updatable for new instances of existing devices, as well as new types of devices.

For guidelines to follow when including CIDS with your application, see [Chapter 33, “Using Common Incremental Device Support”](#). CIDS is functionally dependent on the packages shown in [Table 29-5](#).

CISCO CONFIDENTIAL**Table 29-5** CIDS Package Dependencies

Package Name	Description
CIDS	Common Incremental Device Support
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjgl/jgl	ObjectSpace JGL Classes
CSCOlg4j/log4j	Log4j Logging Framework
CSCOsnmp/snmp	Java SNMP APIs
CSCOWeb/web	Web Services: Apache, OpenSSL, ModSSL

About CWCS Licensing

The CWCS Licensing API allows your application to:

- Install and update licenses
- Create, manage, retrieve and restore license information
- Access FLEXlm license utilities.

The API supports:

- Evaluation-onlylicensing
- Full-purchase licenses.
- Resource-limited licenses, which specify (for example) the number of devices an application can manage.
- Feature licenses, which grant the right to use features within an application.
- Temporary use of a PIN to validate use of a feature.

The API does not explicitly support other kinds of license models, such as node-locked licenses, floating licenses, and counted licenses. However, the API provides access to the FLEXlm toolkit for applications that must implement a licensing model with these kinds of special requirements.

The API provides:

- A repository to store PINs and PAKs.
- An API to install licenses and to retrieve license information, including PIN/PAK.
- A license administration GUI.
- The FLEXlm license toolkit.
- A license backup and restore function.

For guidelines to follow when including the Licensing API with your application, see [Chapter 34, “Using the Licensing APIs”](#). The Licensing API is functionally dependent on the packages shown in [Table 29-6](#).

Table 29-6 Licensing API Dependencies

Package Name	Description
CSCOapch/apache	Core Apache Web Server with SSL
CSCOchlp/chlp	Core Help Files

CISCO CONFIDENTIAL**Table 29-6 Licensing API Dependencies (continued)**

Package Name	Description
CSCOcore/core	Core Modules
CSCOcsdb/ccsdb	Core Database
CSCOess/ess	Event Services Software (includes Tibco event bus)
CSCOjava/java	Core JRE 1.3.1 JARs
CSCOjre2/jre2	CMF JRE 1.2.2
CSCOjrun/jrun	JRun Servlet Engine
CSCOmmd/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOsjre/sunjre	Core JRE 1.3.1 libraries (.so, .font, etc.)
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxrcs/xerces	Apache Xerces XML Parser
SNMPv3 support	SNMPv3 Support for AuthNoPriv Mode
SVC	NT Services (includes TFTP, RSH/RCP, CRM Logger, Blat mail for NT)

About the User Interface Infrastructure

The User Interface Infrastructure (UII) is a web-application framework for creating application GUIs that conform to Cisco's User Experience Initiative (UE) guidelines (available at <http://picasso>). Based on the Apache/Jakarta STRUTS framework, the UII provides:

- A complete set of tools for creating displays, dialogs, tables, lists, wizards, buttons and icons.
- A Reporting Framework that allows you to create and display all types of reports as web pages.
- A Graph Framework that allows you to create and display line, pie, bar, stacked bar, and area graphs.

The UII is not documented in this *Guide*. Guidelines for using these tools and implementing a UII-compliant GUI are provided in the *SDK Developer's Guide for UI Infrastructure* (release 6.1 or later). This document is available

- As a PDF file within the SDK WAR file, or within EDCS (Release 6.1 of the Guide is available as EDCS-275335). You can use the PDF to print a hardcopy version.
- As HTML-based online help. The HTML version is available from the User Experience web site at <http://picasso>. Select **Technology > UII Releases**.

The UII is functionally dependent on the packages shown in [Table 29-7](#). For a working model of the UII, see <http://uii.cisco.com>.

Table 29-7 UII Package Dependencies

Package Name	Description
CSCOgrid/grid	Grid
CSCOhelpDM/cdone	CMF Help Files

CISCO CONFIDENTIAL**Table 29-7** *UII Package Dependencies (continued)*

Package Name	Description
CSCOjawt/jawt	JSCAPE JavaAWT for widgetd
CSCOjcht/jchart	Jchart Java Class
CSCOjpwr/jpwr	JSCAPE Power Search Classes
CSCOjrun/jrun	JRun Servlet Engine
CSCOmd/dmgt	Daemon Manager (Process Manager)
CSCOperl/perl	Perl Support
CSCOswng2/swng2	Java Swing2 Package
CSCOmct/tomcat	Tomcat Servlet Engine
CSCOweb/web	Web Services: Apache, OpenSSL, ModSSL
CSCOxln/xalan	Apache Xalan XSLT Processor
CSCOxml4j/xml4j	IBM XML4J Parser
CSCOxrcs/xerces	Apache Xerces XML Parser
CSCOxsl/lotusxsl	Lotus XSL Engine Classes
GS	Graphing Service
UII	User Interface Infrastructure



CISCO CONFIDENTIAL

CHAPTER 30

Using Object Grouping Services

The CWCS Object Grouping Service (OGS) provides a generic means for creating, managing and sharing groups of objects, regardless of type.

The following topics describe the OGS and how to use it in your applications:

- [Understanding OGS](#)
- [Implementing OGS Servers](#)
- [Creating OGS ASAs](#)
- [Creating an OGS GUI](#)
- [Using OGS Secure Views](#)
- [Using OGS Common and Shared Groups](#)
- [Using OGS 1.3 Client Side Enhancements](#)
- [Using OGS 1.4 Enhancements](#)

For more information about OGS, refer to the following resources:

- *OGS System Functional Specification*, ENG-101932
- *OGS GUI Client Functional Specification*, EDCS-120040
- *OGS GUI Client Software Unit Design Specification*, EDCS-152942
- *Triveni OGS ASA Specification*, EDCS-161116
- *Secure Views and Common Groups In Object Grouping Service Functional Specification*, EDCS-298617
- *Shared Groups in OGS*, EDCS-341632
- *Functional Specifications related to Client side changes in OGS 1.3*, EDCS-347045
- *OGS 1.3 Client related Software Design Specification*, EDCS-358505
- *Kilner Virtual ASA Functional Specification*, ENG-203936
- *Object Selector User Guide*, EDCS-158538
- *SDK Developer's Guide for UII, Release 6.1*, EDCS-275335

CISCO CONFIDENTIAL

Understanding OGS

OGS allows applications to create, manage, and share persistent groups of objects.

OGS is a generic grouping service. It does not supply you with predefined groups. Instead, it provides tools that allow you to define groups useful to your application. Once you have defined the groups you want, you can supply them in predefined form with your application.

OGS places no limits on the types of objects you can group. Most developers use OGS to group network devices. However, you can also use it to manage groups of scheduled jobs, policies, users, tasks, VLANs, subnets, IP phones, user interface views, fault conditions, or any other kind of object that can be grouped based on shared attributes.

The following topics explain the basics of OGS:

- [About the OGS Components](#)
- [Basic OGS Concepts](#)
- [About OGS Groups](#)
- [About OGS Group Types](#)
- [About OGS Container Groups](#)
- [About OGS Group Hierarchy](#)
- [How Rules Are Constructed](#)
- [Choosing to Implement OGS](#)

About the OGS Components

A complete OGS system includes the following components

- **OGS Server:** Manages groups of objects to be shared by applications. For details about implementing this component, see the [“Implementing OGS Servers” section on page 30-7](#).
- **Application Service Adapters (ASAs):** Application-specific processes that serve as sources of the Objects that are grouped by the OGS Server. For details on creating ASAs for your applications, see the [“Creating OGS ASAs” section on page 30-17](#).
- **OGS Clients:** Applications that use the OGS Server to create and manipulate groups in order to perform an application function. You can display OGS data and coordinate OGS interactions using the Object Selector in the GUI for your client application. For details, see the [“Creating an OGS GUI” section on page 30-40](#).

Basic OGS Concepts

OGS makes use of several concepts whose definitions differ from the commonly used definitions:

- **OGS Class:**

An OGS class describes the representation of a set of application abstractions. This differs from traditional object-oriented systems, where a class defines a set of attributes and the operations that can be performed on instances of that class.

CISCO CONFIDENTIAL

An OGS class is concerned only with the representation and retrieval of attributes. Class hierarchies are supported and are interpreted in the conventional sense; that is, a subclass inherits all the attributes of its superclasses. An instance of an OGS class can be used in any context where an instance of one of its superclasses is expected.

- **OGS Object:**

An OGS object is a collection of attribute values. The OGS class of an object determines the set of attributes associated with that object. Associated with every object is a unique and immutable object ID (OID). When an ASA evaluates a rule, the data returned to the OGS Server is a collection of proxy objects that contains:

- The OIDs of the grouped objects.
- Any attributes of those objects that were requested.

These proxy objects are referred to as OGS objects in the rest of this document.

- **OGS Group:**

An OGS group is a named aggregate entity comprising a set of objects belonging to a single class or a set of classes with a common superclass. Groups can be shared between users or applications, subject to access-control restrictions. The membership of a group is determined by a rule.

- **OGS Rule:**

An OGS rule consists of one or more rule expressions combined by operators, which can be AND, OR or EXCLUDE. A rule always evaluates to objects of a particular class defined in an application schema.

About OGS Groups

A group in OGS is a named aggregate entity comprised of a set of objects. Each group has:

- A set of properties, such as group ID, name, description, permission, etc.
- An associated rule. The rule determines the members of a group, which may change whenever the rule is evaluated.

OGS Servers represent the group membership as a list of object IDs. Besides the OID list, attributes associated with the objects in a group may also be queried. The attributes of a class and the hierarchical relationship of classes are defined in the Application Service Adaptor schema, which must be registered with the OGS.

Each OGS group has a set of properties, including a unique name. [Table 30-1](#) describes these properties.

Table 30-1 OGS Group Properties

Property	Description
GroupID	A unique identifier for the group.
GroupName	Fully qualified name of the group. This name must include the parent's path, with each element of the path being separated by the / character.
Description	User-specified text description of the group.
Created By	ID of the user who created the group.
Created Time	Date and time the group was created.
Last Modified By	ID of the user who last made changes to the group.
Last Modified Time	Date and time of last modification.

CISCO CONFIDENTIAL

Table 30-1 OGS Group Properties

Property	Description
Last Evaluated By	ID of the user who last evaluated the group.
Last Evaluated Time	Date and time of last group evaluation.
Type	Static or Dynamic.
Access Control	Permission list to read/write this group.
Rule	Rule to determine memberships of the group.
Tags	Variable list of name-value pairs defined by the user.

About OGS Group Types

OGS groups can be either dynamic or static (see also the [“About OGS Container Groups”](#) section on page 30-5).

A dynamic group is one whose membership list is effectively computed every time a user views its members. For each request to view the group, the OGS Server may automatically request the relevant Application Service Adapter to recompute the group's rule. The Server is not required to store a dynamic group's membership list (although it may cache the results of the last re-evaluation), and viewing a dynamic group will always give the latest group membership.

Example

A user creates the dynamic group “MyDevices”, which has the rule "IPAddress in Subnet 172.20.32.0/24". At creation time, the rule evaluates to devices D1, D2, and D3. If a user attempts to view the group membership at any later time, the OGS Server will return the current membership.

If new devices D4 and D5 joined subnetwork 172.20.32.0/24 in between the time the group was created and the time its membership was queried, OGS will return devices D1, D2, D3, D4 and D5 as the current members of the group.

A static group is one whose membership is refreshed only when a user explicitly requests it. Between re-evaluations, the OGS Server stores the static group's membership list and group definition. Whenever a user views a static group, OGS returns the membership list the ASA created the last time the group rule was evaluated.

Note that the OGS may cache the membership and maintain the cache by other means. Group definitions are always stored, regardless of whether the group is static or dynamic.

Example

A user creates the static group “MyDevices”, which has the rule "IPAddress in Subnet 172.20.32.0/24". The rule evaluates to devices D1, D2, and D3, and the OIDs for these devices is stored along with the group definition.

If a user attempts to view the group, he will see only devices D1, D2 and D3, as these were the devices that satisfied the group rule when the group was created.

If the user wants to refresh the membership of the group, then he must explicitly request it. The OGS Server will then ask the ASA to reevaluate the rule.

If new devices D4 and D5 joined subnetwork 172.20.32.0/24 in between the time the group was created and the time the user requested the refresh, OGS will return devices D1, D2, D3, D4 and D5 as the current members of the group. This device list will also be stored as the membership of “MyDevices” until the next refresh.

CISCO CONFIDENTIAL

Most OGS groups are dynamic. Static groups are useful when you want users to be able to “snapshot” a group’s membership and maintain it until a later time, when the user can quickly update it without having to change the group definition.

Example

A network administrator in a large enterprise is responsible for the management of all operational Catalyst 6K series switches. Switches in this enterprise must go through several configuration stages before they are considered operational. Configuration is done by a department to which the Network Administrator does not belong. Before becoming operational, the switches may also be available on the network.

To cope with this, the Network Administrator could create a static group called "My Cat6K Devices" with the rule "DeviceType == Catalyst 6K". This would allow him to accomplish the following:

- The group "My Cat6k Devices" would be populated with the operational Catalyst 6K devices in the network at the time.
- The enterprise acquires new Catalyst 6K devices, deploys them in the network, and begins configuring them. The administrator continues to use the group "My Cat6k Devices" with the assurance that the new Catalyst 6K devices will be excluded from the group.
- When the new switches are operational, the administrator requests a refresh of the group without making changes to the group rule. “MyCat6K Devices” now includes the new switches.

About OGS Container Groups

Container groups are a separate type of group, on par with normal groups. This is because they are groups who have no membership of their own. A container group is an “empty” container, whose membership is simply the union of the membership of all its subgroups.

This is different from normal groups that must have a rule to determine its membership. Instead, a container group’s effective rule consists of:

- If it has no subgroups: No rule.
- If it has only one subgroup: The same effective rule as its subgroup.
- If it has more than one subgroup: A rule composed of the effective rules of all its subgroups, combined with the operator OR.

Container groups can be static or dynamic depending on the behavior of their subgroups:

- Dynamic Container: Its membership is the aggregate membership of its subgroups at any given time.
- Static Container: Its membership (as the aggregate of its subgroup memberships) is recomputed only when new subgroups are added, existing subgroups are deleted, when the subgroups’ rules are modified, or upon request.

About OGS Group Hierarchy

OGS manages groups in a hierarchical fashion and supports subgrouping. Each child group is a subgroup of a parent group and its group membership will be a subset of its parent group. OGS also supports container groups. Container groups are groups with no rule, whose membership is simply the union of the membership of its children.

Regardless of whether a group's parent is static or dynamic, the result of evaluating a group is the intersection of the objects that satisfy its rule and the objects that satisfy its parent's effective rule.

CISCO CONFIDENTIAL

The effective rule for a static group is a rule that enumerates its members. When a static group is reevaluated, all its descendant static groups will also be reevaluated.

The effective rule for a dynamic group is an expression that is formed by applying the operator AND to the operands that are the group's rule and its parent's effective rule.

How Rules Are Constructed

A group rule consists of one or more expressions, which can be combined using the operators AND, OR or NOT. Each expression has the following components, concatenated with a ":" separator

- Domain: The cluster of applications sharing the group (e.g., "Triveni" or "ALL").
- Application: The specific application within the domain that shares the group (e.g., "RME" or "Campus")
- Class: The class of object (e.g., "Device").
- Attribute: The specific object attribute whose value the rule will test (e.g., "DeviceType" or "IPAddress"),
- Operator: The evaluation operation defined in the Application Schema (e.g., "IsEqualTo" or "Contains").
- Value: The value to be tested for (e.g., "Router").

Example

The following rule will select Devices which are either Routers or have an IOS Version greater than 11.3 and that are assigned IP Addresses beginning with the octets "172.20":

```
MYAPP:RME:Device.DeviceTypeEquals "Router" or MYAPP:RME:Device.IOSVersion > "11.3" AND  
MYAPP:RME:Device.IPAddressContains "172.20"
```

Choosing to Implement OGS

If you want to implement OGS in your application without extensive customization, you must also:

- Install the Common Services Transport Mechanism (CSTM). The OGS Server uses CSTM to communicate with its clients. For more information on CSTM, see [Chapter 31, "Using the Common Services Transport Mechanism."](#)
- Implement a local database. The OGS Server persists group information to a local data store to ensure that group information is not lost between activations. Your choice of DBMS to accomplish this should adhere to any guideline or prescription established for your application. For information on implementing a database in Common Services, see [Chapter 11, "Using the Database APIs."](#)
- Implement Event Services Software (ESS), including the Java Messaging Service (JMS). For more information on these topics, see [Chapter 19, "Using Event Services Software."](#)
- Implement the OGS Server. For more information on this task, see the ["Implementing OGS Servers" section on page 30-7.](#)
- Create an OGS Application Service Adapter and Application Schema File for your application. For more information on this task, see the ["Creating OGS ASAs" section on page 30-17.](#)

In addition to these requirements, your application must support JDK 1.3.1.

CISCO CONFIDENTIAL

Implementing OGS Servers

The OGS Server performs the following tasks:

- Creating and maintaining group information.
- Interacting with Application Service Adapters (ASAs) to:
 - Evaluate group membership.
 - Retrieve the requested attributes of the member objects of a group.
 - A later release will allow ASAs to register their schema. Currently, you must register ASA schema statically, via configuration files.

The OGS Server provides its clients with a unified and consistent view of group definitions and results of rule evaluation. It does this by recording any change to the group information (through creation, modification, deletion or evaluation of a group).

The OGS Server relies on:

- A means of persisting the group definition and membership data. This can be a relational database, object-oriented database, or file system.
- The Common Services Transport Mechanism (CSTM) to interact with clients.
- ASAs to perform rule evaluation.

The following topics discuss OGS Server and its implementation in detail:

- [Getting Started with OGS Server](#)
- [How OGS Server Works](#)
- [Using the OGS Server APIs](#)
- [Customizing OGS Server Interfaces](#)
- [Creating a Custom OGS Event Processor](#)
- [Handling OGS Exceptions](#)

Getting Started with OGS Server

Setting up a running OGS Server instance normally requires you to create an Application Service Adaptor (ASA) for your application and then set the OGS Server to use it.

You can also create a very simple OGS Server implementation using the test ASA supplied on the SDK. The following steps explain how to install OGS Server and configure it to use TestASA through a script file.

For details on writing an ASA that performs work useful to your application, see [“Creating OGS ASAs” section on page 30-17](#).

-
- Step 1** Retrieve the packaged version of the OGS WAR file from the [OGS Portal](#).
- Step 2** Extract the WAR file’s contents to the default application directories under Tomcat. For example:
- Extract all OGS jar files to `/tomcat/webapps/appname/WEB_INF/lib` (where *appname* is the name of your application).
 - Extract all other files (in including property and configuration files) to `/tomcat/webapps/appname/WEB_INF/classes`.

CISCO CONFIDENTIAL

- Step 3** Retrieve TestASA.tar (or TestASA.zip) from the SDK portal.
- Step 4** Extract TestASA.tar (or TestASA.zip) to the default application directories under Tomcat.
- Step 5** Run the OGS Server test setup script ServerSetup.sh (ServerSetup.cmd). The script will update OGSServer.properties and ASARegistry files to point to TestASA, exactly as you would do when implementing your own custom ASA (see [“Running a Customized ASA” section on page 30-36](#)).
- Step 6** Start CWCS and OGS Server. Use the Administrative GUI to add some groups.
-

How OGS Server Works

OGS Server makes use of five classes to instantiate its most important behaviors:

- **OGSClassDefinition:**
This class models the OGS notion of classes (see the [“Basic OGS Concepts” section on page 30-2](#)). The OGS Server creates instances of OGSClassDefinition using the definitions of OGS classes in an application schema. An OGSClass instance holds information about an application class, such as its name, its domain and the application to which it belongs, the attributes associated with it, attributes that can be used in composing rules, the operations that are permitted on those attributes, and the allowable target values.
- **OGSObject:**
This class models the OGS notion of objects (see the [“Basic OGS Concepts” section on page 30-2](#)). ASAs create instances of OGSObject and return lists of them to the OGS Server whenever the ASAs evaluate rules. Each OGSObject instance contains the OID of the application class object it represents, and the values of the other attributes that were requested by a client.
- **OGSRule, OGSRuleExpression:**
These two classes model the expressions used in rules and the rules themselves.
- **OGSGroupDefinition:**
This class models OGS groups in a straightforward manner. It has instance variables that store group information, such as group ID, name, ownership, creation and modification times, and so on.

An active OGS Server instance is an executing operating system process that runs as a daemon. Developers should use the CWCS Daemon Manager (see [Chapter 17, “Using the Daemon Manager”](#)) to manage OGS Server states.

Upon activation, an OGS Server process performs initialization tasks, such as reading static ASA registration and group definitions from its persistent store.

If initialization is completed successfully, OGS Server publishes its URN to CSTM and waits for requests. When it receives requests, it passes them to the appropriate registered ASA, and returns the result.

Using the OGS Server APIs

The OGS Server API allows you to:

- Create, delete, copy, modify and rename groups
- Copy group hierarchies
- Evaluate the members of a group

CISCO CONFIDENTIAL

- Retrieve application schema and class information
- Retrieve group definitions
- Retrieve lists of all group names
- Verify service

Usage, input arguments and other details for each OGSServer API call are fully documented in the OGS Javadoc available on the [OGS Portal](https://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=6625) at https://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=6625.

The following example demonstrates how to implement the API as part of a client application.

Example 30-1 A Sample OGS Client

```

package com.cisco.nm.xmls.ogs.client;

import java.util.ArrayList;
import com.cisco.nm.xmls.ogs.server.OGSInterface;
import com.cisco.nm.xmls.ogs.util.OGSoid;
import com.cisco.nm.xmls.ogs.util.OGSSecurityContext;
import com.cisco.nm.xmls.ogs.util.OGSObjectList;
import com.cisco.nm.xmls.ogs.util.OGSObject;
import com.cisco.nm.xmls.ogs.util.ClassPath;
import com.cisco.nm.xmls.ogs.client.OGSServerProxy;

public class GetDeviceGroups {

    // This program retrieves all groups managed by an OGS Server
    // that have at least one device object as a member. The program
    // is invoked with a single argument that names the class that
    // represents device objects.

    public static void main(String[] args)
    {
        if (args.length != 1) {
            System.out.println("usage: GetDeviceGroups device-class-name");
            System.exit(-1);
        }
        _devClassName = args[0];
        _devClassLength = _devClassName.length();
        try {
            _devClassElements = new ClassPath(args[0]).elements();
        } catch (Exception ex0) {
            System.err.println("Error getting elements of class: " + args[0]
                + ": " + ex0.getMessage());
        }
        TaskID task=new TaskID ("OGS", "OGSOPERATION", null); OGSSecurityContext ctxt=new
OGSSecurityContext(_adminName,task);
        try {
            // Get the hierarchical membership of the root of all
            // groups. Having retrieved the membership in this
            // fashion, we can now examine the membership of
            // all the descendant groups and determine the ones
            // whose membership contains at least one device object.

            OGSServerProxy pxy = new OGSServerProxy();
            Integer mtype = new Integer(OGSInterface.HIERARCHICAL_MEMBERSHIP);
            OGSObjectList list
                = pxy.evaluateGroup(ctxt,
                    null,
                    null,

```

CISCO CONFIDENTIAL

```

        _root,
        null,
        mtype,
        Boolean.FALSE);

    String[] devices = getDeviceGroups(list);
    for (int i = 0; i < devices.length; ++i) {
        System.out.println(devices[i]);
    }
} catch (Exception ex) {
    System.err.println("Error getting device groups: " +
        ex.getMessage());
}
}

private static String[] getDeviceGroups(OGSObjectList list)
{
    ArrayList result = new ArrayList();
    OGSObjectList[] children = list.children();
    for (int i = 0; i < children.length; ++i) {
        getDeviceGroups(list, result);
    }
    return (String[])result.toArray(new String[0]);
}

// This method gathers the names of all the groups
// (in the hierarchy rooted at the group represented by 'list')
// have at least one device object as a member.

private static boolean getDeviceGroups(OGSObjectList list, ArrayList result)
{
    boolean hasDevice = false;
    OGSObjectList[] children = list.children();
    for (int j = 0; j < children.length; ++j) {
        if (getDeviceGroups(children[j], result)) {
            hasDevice = true;
        }
    }
    // If any descendant has a device member, this group
    // has it too.

    if (hasDevice) {
        result.add(list.name());
        return true;
    }

    // This group has no descendants that have a device
    // member, so check the membership of the group for
    // devices.

    OGSObject[] objects = list.objects();
    for (int i = 0; i < objects.length; ++i) {
        String oid = objects[i].objectId();
        try {
            String objClass = OGSoid.getClassFromOid(oid);
            if (isDeviceClass(objClass)) {
                result.add(list.name());
                return true;
            }
        } catch (Exception ex) {
            System.err.println("Error processing object: " + oid);
        }
    }
    return false;
}

```

CISCO CONFIDENTIAL

```

    }

    // Returns true if the class named by 'cname' is the same
    // as the device class or is a subclass.

    private static boolean isDeviceClass(String cname)
    {
        if (cname.length() < _devClassLength) {
            return false;
        } else if (cname.equals(_devClassName)) {
            return true;
        }

        // The following can also be achieved by verifying that
        // _devClassName is a prefix of cname and that the character
        // following the matching prefix is ':'.

        try {
            ArrayList classElements = new ClassPath(cname).elements();
            if (classElements.size() < _devClassElements.size()) {
                return false;
            }
            for (int i = 0; i < _devClassElements.size(); ++i) {
                String s1 = (String)classElements.get(i);
                String s2 = (String)_devClassElements.get(i);
                if (!s1.equals(s2)) {
                    return false;
                }
            }
            return true;
        } catch (Exception ex) {
            System.err.println("Error getting elements of: " + cname);
            return false;
        }
    }

    private static int _devClassLength;
    private static String _devClassName;
    private static ArrayList _devClassElements;
    private static String _root = "/";
    private static String _adminName = "admin";
}

```

**Note**

This code is in the OGS VOB and can be found at:
 /vob/enm_ogs/share/classes/client/com/cisco/nm/xms/ogs/client/OGSClient.java

Customizing OGS Server Interfaces

All OGS Server interfaces that you can customize are listed as settable parameters in the OGSServer.properties file. These customizable interfaces include:

- **CacheImplClass**

This is an implementation of the interface OGSGroupCacheIf (see [Figure 30-1](#)). OGS uses an instance of this class as a source of grouping data. Though an ASA ultimately evaluates the rule, this abstraction is defined so that group membership can be cached. The OGS Server uses an instance of

CISCO CONFIDENTIAL

the class specified for this property for every ASA that is registered. A limitation in this release of OGS is that instances of only a single class can be used for caching, even when multiple ASAs are used (that is, the caching strategy cannot be customized for the individual ASAs). The default value is `com.cisco.nm.xml.ogs.server.GroupCacheImpl`. Use the default implementation if possible, or subclass `GroupCacheImpl` (as Kilner does).

- **GroupPersistorImplClass**

An implementation of the interface `OGSGroupPersistorIf` (see [Figure 30-1](#)). The OGS Server uses this class to persist and retrieve group definitions. The default value for this property is `com.cisco.nm.xml.ogs.server.GroupPersistorImpl`. This implementation uses the CWCS database to persist group definitions. Use the default implementation unless your product has special requirements.

- **GroupCachePersistorImplClass**

This is an implementation of the interface `GroupCachePersistorIf` (see [Figure 30-2](#) and the caching information in this topic). Instances of `GroupCacheImpl` use an instance of this class to persist cached membership data, so this property is relevant if the OGS is configured to use class `GroupCacheImpl` (or its subclass) only.

The default value for this property is `com.cisco.nm.xml.ogs.server.GroupCachePersistorImpl`. Use the default implementation unless your product has special requirements.

- **SecurityImplClass**

This is an implementation of `OGSSecurityHandlerIf` (see [Figure 30-1](#)). OGS uses an instance of this class to enforce access control on groups, but not on the group membership.

The default value for this property is `com.cisco.nm.xml.ogs.server.DummySecurityHandler`, which allows unrestricted access by all users. You can find examples of implementations that follow different security requirements (those for Campus Manager and Kilner) in the OGS VOB.

- **ASARegistrationFile**

This specifies the XML file used to register ASAs with the OGS Server and identifies the ASA implementation to be used. See the “[Registering the ASA with OGS](#)” section on [page 30-36](#) for an example ASA Registration File and its DTD.

- **SystemGroupsFile**

The file that contains the definitions of any predefined groups you have customized for your product and that you will ship with OGS. For a sample version of this, see Kilner’s System Groups file, `vasa-system-groups.xml`.

- **SystemGroupCreator**

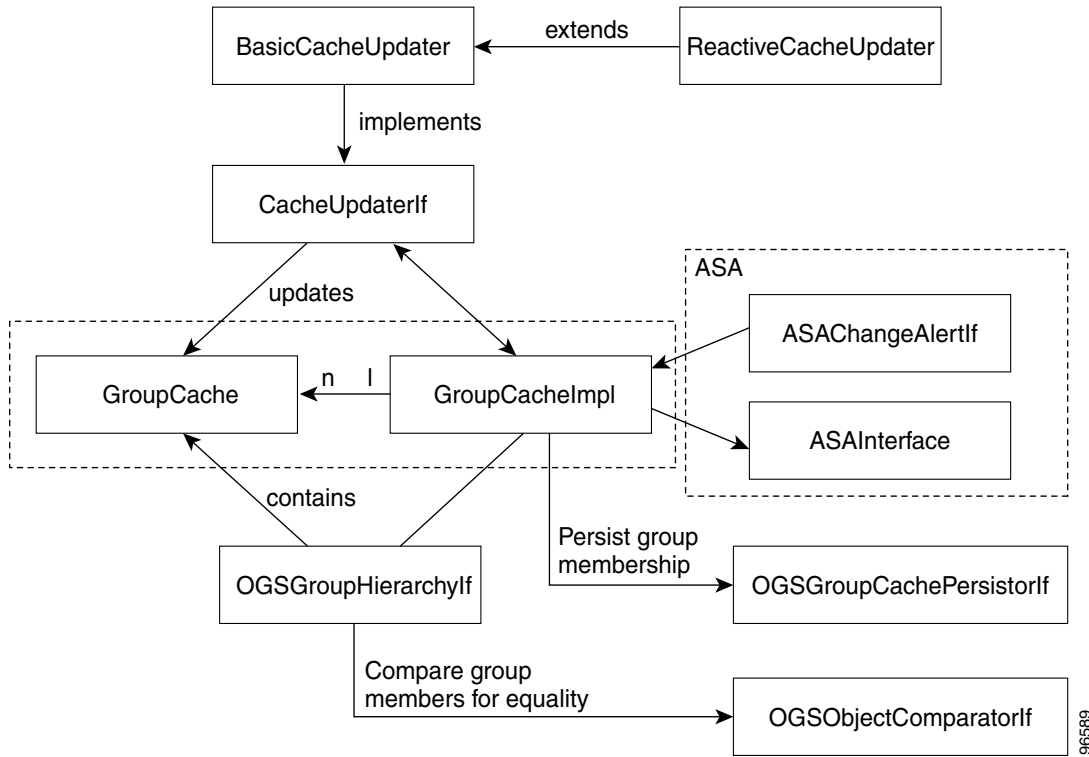
This specifies the name of the creator of the system predefined groups given in the `SystemGroupsFile`. This can be any string, but your security module may place some restrictions on what this should be. The default value is “ogs”.

- **OGSEnvironmentAdapterImpl**

This class can be used to do any postprocessing operation after the `OGSServer` starts in the environment.

For example, the published URNs need to be unpublished URNs need to be unpublished when the daemon manager stops the OGS server. This is done by the implementation of `OGSEnvironmentAdapterIf` interface. Another related property is `OGSEnvRegistrationName` which specifies the process or daemon name of the OGS server.

[Figure 30-1](#) shows the relationships among these OGS Server properties.

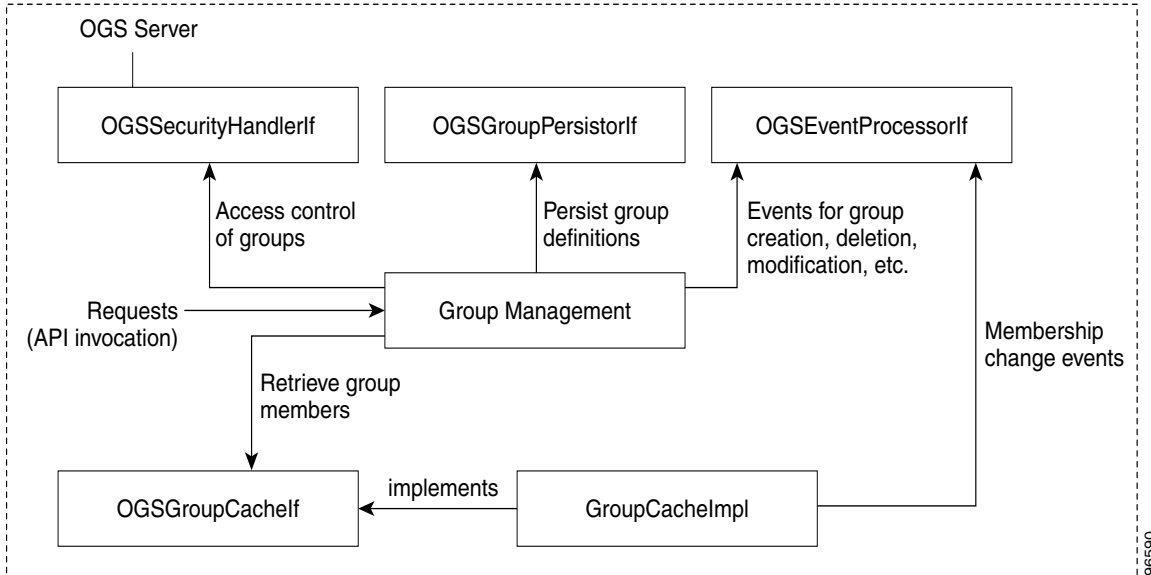
CISCO CONFIDENTIAL**Figure 30-1** OGS Server Interfaces

In addition to these interfaces, the default OGS Cache Manager (class `GroupCacheImpl`) has three cache update mechanisms that it activates periodically. All three update mechanisms use implementations of the interface `CacheUpdaterIf` shown in [Figure 30-2](#):

1. **BasicCacheUpdater**—Updates the cached membership by evaluating all the groups.
2. **ReactiveCacheUpdater**—Collects change information (such as objects added or deleted, attribute value changes, etc.) and updates the cached membership of the groups affected by those changes.
3. **InvalidCacheUpdater**—Updates cached membership by evaluating groups for which the previous evaluation had failed.

CISCO CONFIDENTIAL

Figure 30-2 OGS Cache Manager Interfaces



Of these three updater mechanisms, the first two (**BasicCacheUpdater** and **ReactiveCacheUpdater**) can be customized. To customize the two updaters, you can set the following **OGSServer.properties** parameters:

- **GroupUpdateFrequency**
Sets the interval in seconds between activations of the **BasicCacheUpdater**. A value of -1 disables this updater. The default value is 60.
- **GroupUpdaterClass**
TBD
- **ReactiveUpdaterClass**
Specifies the name of the class that implements **CacheUpdaterIf** (see [Figure 30-2](#)). This is normally the default class or one of its subclasses: **ReactiveCacheUpdater** in `com.cisco.nm.xms.ogs.server`. For an example of a custom reactive updater, see **KilnerReactiveUpdater** in `com.cisco.nm.xms.ogs.Kilner10`.
- **ReactiveUpdateFrequency**
Sets the interval in seconds between activations of the reactive updater. A value of -1 disables this updater. The default value is -1 (that is, the reactive updater is disabled in the default configuration).

OGSServer.properties also allows you to customize the following properties:

- **ProductId**
Sets the string representing the ID of the product in which this OGS implementation is used. This is used to create strings for names of events published by this instance of OGS.
- **InstanceId**
Sets the string representing this instance of OGS. This is reserved for future use.
- **IgnoreCacheInitErrors**

CISCO CONFIDENTIAL

Specifies whether you want cache initialization errors to be ignored when creating dynamic groups, so that dynamic group creation will not fail. This property does not affect static groups, since these groups, by definition, require that their membership be completely determined at creation time. We recommend that you set this property value to True; the default value is False.

- **EventProcessorImplClass**

Specifies the name of the class that implements `OGSEventProcessorIf` (see [Figure 30-2](#)). This interface allows a product to customize how OGS events are dispatched. Normally, this is the default class `DefaultEventProcessor`, which dispatches OGS events with no modifications.

For an example of a custom Event Processor, see `KilnerEventConsolidator` in `com.cisco.nm.xml.ogs.Kilner10` and the [“Creating a Custom OGS Event Processor”](#) section on [page 30-15](#), which examines `KilnerEventConsolidator` in detail.

Creating a Custom OGS Event Processor

The `DefaultEventProcessor` in OGS dispatches OGS events without modifications. Each such event is dispatched as a message, with a subject name that looks like this:

```
cisco.mgmt.cw.product_id.ip_address.ogs.alert
```

Where:

- `product_id` is the name of the product publishing the event.
- `ip_address` is the IP of the server where OGS is executing.

The subject of the message describes an atomic change to the data OGS manages, including group creation, deletion, modification, renaming, or membership change. This can often result in hundreds of messages.

For example, consider a group high in a group hierarchy that is deleted. In this case, atomic “deletion” messages will be generated for every subgroup. If there are 50 such subgroups, the default Event Processor will forward 51 “group deletion” messages. Similarly, a single modification in a group containing 1000 objects would result in 1000 group-membership change events.

You can create your own Event Processor, and use the `OGSServer.properties` parameter “`EventProcessorImplClass`” to implement it (see the [“Customizing OGS Server Interfaces”](#) section on [page 30-11](#)). The custom Event Processor can handle events in any way you wish, up to and including suppressing all events.

Kilner offers an example of a custom Event Processor: `KilnerEventConsolidator` in `com.cisco.nm.xml.ogs.Kilner10`. This processor dispatches customized “consolidated events”, which logically group or aggregate atomic OGS event information based on the operation that triggered those events.

The subject name for a consolidated event in Kilner looks like this:

```
cisco.mgmt.cw.kilner.ip_address.ogs.alert.consolidatedevent
```

Where `consolidatedevent` is a single message that consolidates the individual events that are the results of a single “groupcreation”, “groupdeletion”, “groupmodification”, “grouprename”, “membershipchange”, or “serverstart” operation. The body of the message will contain the atomic event data consolidated under that “heading”.

For every consolidated event, an instance of class `ConsolidatedOGSEvent` in `com.cisco.nm.xml.ogs.kilner10` is sent in an object message.

CISCO CONFIDENTIAL

Processing requires:

1. Using the method ogsOperation() to retrieve the OGS operation that resulted in the event. The relevant OGS operations (defined in class ConsolidatedOGSEvent) are:
 - Group Creation: GROUP_CREATION, GROUP_COPY, GROUP_HIERARCHY_COPY, CREATE_PARTITION
 - Group Deletion: GROUP_DELETION
 - Group Rename: GROUP_RENAME
2. If the event is of interest, retrieving and processing the collection of associated AtomicEventData.

For more on creating a consolidated Event Processor, see:

- The Kilner VOB.
- The ConsolidatedOGSEvent and AtomicEventData Javadoc at the [OGS Portal](https://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=6625), https://mco.cisco.com/ubiapps/portal/go.jsp?portal_id=6625.

Handling OGS Exceptions

The OGS API defines the exception classes shown in Table 30-2. The list includes a description of the kinds of operations that typically result in these exceptions. All of these exception classes are subclasses of OGSException. In addition to these exceptions, you should be sure that your application is prepared to handle instances of OGSException that represent internal errors.

Table 30-2 *OGSException Classes*

Exception	Description	Operations
OGSException.AccessDenied	The user does not have the access required to perform the requested operation.	All operations except retrieveApplicationSchema, validateRule and verifyService.
OGSException.GroupExists	The named group already exists.	createGroup, copyGroup, copyGroupHierarchy, renameGroup
OGSException.GroupDoesNotExist	The named group does not exist.	All operations except retrieveApplicationSchema, validateRule, createGroup, retrieveGroupList, verifyService.
OGSException.InvalidParent	The parent group, as specified in the name of a group, is invalid (does not exist).	createGroup, copyGroup, copyGroupHierarchy
OGSException.IllegalRename	The new name provided while renaming a group is not legal. OGS does not allow renaming a group such that its parent group is changed. For example: This exception will be thrown if you attempt to rename group /A/B/C to /A/D/C1.	renameGroup
OGSException.IllegalContainer	A process attempted to create a container group (a group with no rule) under the root.	createGroup, copyGroup, copyGroupHierarchy

CISCO CONFIDENTIAL**Table 30-2** *OGSException Classes (continued)*

Exception	Description	Operations
OGSException.IllegalName	The group or class name is malformed.	All operations except retrieveApplicationSchema, retrieveGroupList, verifyService.
OGSException.IllegalGroupDefinition	Thrown whenever the definition of a group is malformed. Currently, the only condition that will provoke this exception is when the evaluation type is not specified as either static or dynamic.	All constructors of OGSGroupDefinition
OGSException.InvalidRule	The rule is invalid.	createGroup, copyGroup, copyGroupHierarchy, modifyGroup, evaluateRule, validateRule.
OGSException.IllegalNullParameter	One of the parameters being passed has a illegal null value.	All operations except verifyService, retrieveApplicationSchema.
OGSException.IllegalParameter	One of the parameters has an illegal (but not null) value.	evaluateGroup (when the membership format is not one of the specified formats).
OGSException.ASANotFound	The operation could not find an ASA that could evaluate the rule.	createGroup, modifyGroup, validateRule, evaluateRule.
OGSException.GroupInitializationFailed	The initialization of the group's membership failed.	evaluateGroup
OGSException.ASAError	The process received an error from the ASA while evaluating a rule.	createGroup, copyGroup, copyGroupHierarchy, modifyGroup, evaluateRule, evaluateGroup.

Creating OGS ASAs

In order to use OGS, you must create and associate with your application an Application Service Adapter (ASA). The OGS Server depends on ASAs to:

- Respond to queries that seek to verify the presence of an ASA.
- Register an application schema that the OGS Server supports. OGS Servers process schema registration files at startup.
- Evaluate a rule and return to the OGS Server the OGS objects that satisfy it.
- Given a list of object IDs and a set of attributes, return the corresponding OGS objects.

The remainder of this topic explains what is needed to create an OGS ASA, including:

- [Understanding ASA Infrastructure Modules](#)
- [Customizing ASA Infrastructure Modules](#)
- [Running a Customized ASA](#)

CISCO CONFIDENTIAL

About ASA Implementations

There is no default ASA supplied with OGS. However, the OGS team provides a set of libraries and tools to assist in ASA development, and default or example implementations for most modules.

Application developers and the OGS Team have also collaborated in creating OGS ASAs for several applications. These serve as ready-made examples of how to create customized ASAs, and their code is available on a read-only basis to Cisco developers with access to ClearCase:

- For code related to the ASA infrastructure classes (and example implementations, such as the Generic SQL ASA), see the OGS vob, /vob/enm_ogs/share/classes/server. All packages are defined within this vob.
- For the latest code related to the generic OGS, use the views created in project ogs.
- To access code related to Kilner and its ASAs (such as the VMQueryGenerator in the class com.cisco.nm.xml.ogs.kilner.), use the views created in project ogs_kilner.

Managing ASA Processes

OGS Server will load an ASA into its JVM if the ASA Registration File (see the [“Running a Customized ASA” section on page 30-36](#)) specifies a Location type of “local”. OGS Server will manage startup and shutdown, and registration with CSTM, of any local ASA.

Developers should use the CWCS Daemon Manager (also known as Process Manager; see [Chapter 17, “Using the Daemon Manager”](#)) to manage startup and shutdown of the OGS Server.

A “local” ASA is the only ASA type allowed in this release of OGS. Future OGS versions will allow ASAs to run remotely, either alone or within the JVM of a non-OGSServer application.

When planning your application’s use of OGS in future releases, please note that remote ASAs will need to register themselves with OGSServer and with CSTM, and must be controlled by the CWCS Daemon Manager, to ensure they are available to requests from OGS Servers.

Using ASAs to Aggregate Data

Under normal circumstances, OGSServer is responsible for aggregating data resulting from the evaluation of a query on multiple instances of an ASA. However, it is possible to customize ASAs to perform this function if requirements make this necessary.

For example, an ASA may require tight cooperation between its instances to evaluate rules. In this case, the ASA can register a single instance that will then be responsible for communicating client requests to the other instances and aggregating the results before returning them to the OGS Server.

About the OGSServer-ASA Interface

Provided as `ASAInterface` in `com.cisco.nm.xml.ogs.asa`, this component allows the ASA and OGSServer to exchange commands, queries and result data via the Common Services Transport Mechanism.

All current ASA structures implement this interface. However, the interface is pluggable, which means that you can substitute your own ASA structure for the one described here.

You may want to do this if you have a very simple object grouping requirement.

For example, you may require OGS to parse a flat file containing nothing but a list of user names. In this case, you can write your own simple ASA class and implement it. However, your ASA class must:

- Be capable of responding on its own to requests from the OGSServer as normal ASAs do (that is, it must be able to parse rules, access the flat file, etc.).

CISCO CONFIDENTIAL

- Implement `ASAInterface`.
- Be registered with `OGSServer`. You do this by entering the name of this class as the `classname` value in the ASA Registration file (for more on this, see the [“Registering the ASA with OGS”](#) section on page 30-36).

Understanding ASA Infrastructure Modules

The basic function of your application’s ASA is to validate and evaluate rules passed to it by the OGS Server. To perform these functions, your ASA needs the following modules:

1. **Rule Validator:** Ensures that a rule passed to it by the OGS Server is valid. For details, see the [“About the Rule Validator”](#) section on page 30-20.
2. **Generic Schema:** Provides for the Rule Validator a parsed object tree containing all the combinations of groupable classes, attributes, operators and values allowable in rules. For details, see the [“About the Generic Schema”](#) section on page 30-20.
3. **Rule Evaluator:** Converts a validated rule into a query appropriate to the application’s data-storage structure, and returns to the OGS Server the objects that match the rule. For details, see the [“About the Rule Evaluator”](#) section on page 30-21.
4. **Mapping Schema:** Maps groupable classes against the data-storage elements in an application domain and supplies the mapping to the Rule Evaluator for query formation. For details, see the [“Customizing the Mapping Schema”](#) section on page 30-27.
5. **Rule Converter:** Translates the notation for a rule used by the OGS Server into the notation used internally by the ASA infrastructure. For details, see the [“About the Rule Converter”](#) section on page 30-21.
6. **Node Rule Expression:** Models the basic type of Rule Expression that the ASA will validate and evaluate. For details, see the [“About Node Rule Expressions”](#) section on page 30-21.
7. **Composite Node Rule Expression:** Models complex Rule Expressions involving two or more Node Rule Expressions. For details, see the [“About Composite Rule Expressions”](#) section on page 30-22.
8. **ASA Change Alerter:** Alerts the OGS Server when an object changes. For details, see the [“About the ASA Change Alerter”](#) section on page 30-22.

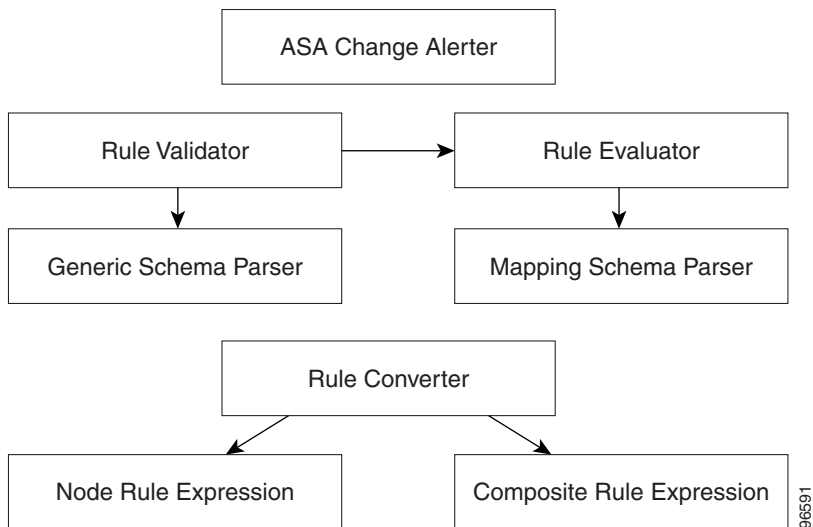
Figure 30-3 shows the relationships among these modules.

For guidelines and examples on how to customize these infrastructure modules for your application, see the [“Customizing ASA Infrastructure Modules”](#) section on page 30-22.

For other examples of custom ASAs, see the ClearCase VOBs listed in the [“Creating OGS ASAs”](#) section on page 30-17.

CISCO CONFIDENTIAL

Figure 30-3 ASA Infrastructure Modules



96591

About the Rule Validator

The Rule Validator ensures that:

1. Each groupable class in the rule is valid.
2. Each groupable attribute in the rule is valid.
3. Each groupable operator in the rule is valid.
4. The rule as a whole evaluates to a valid class. This includes all of the Rule Expressions, combined by AND, OR, or EXCLUDE operators.
5. The object ID values specified in the rule refer to actual objects. The OIDs can contain the value “ANY” or a “\$”- separated value produced by the ASA.

In addition to these tasks, the Rule Validator can also check the validity of any:

- Value range specified in the rule.
- Value enumeration specified in the rule.

The Rule Validator depends on the Generic Schema to perform these tasks.

About the Generic Schema

The Generic Schema consists of a Parser and XML Schema file describing the groupable classes, attributes, operators and values (including any value ranges and enumerations) valid for all objects. The Parser supplies this structure to the Rule Validator.

CISCO CONFIDENTIAL

About the Rule Evaluator

The Rule Evaluator:

1. Evaluates the rule for a particular OGS class.
2. Combines individual Rule Expressions within the rule so that the object data can be retrieved. “Combining” the Rule Expressions can mean turning all of them into one query (as in the SQLQueryGenerator), or evaluating each Rule Expression separately and then combining the results based on the operator (as with the VMSAQueryGenerator).
3. Returns to the OGS Server the objects that match the rule.
4. Defines a unique object ID to represent the OGS object in the domain. For example, in ANI, the database identifier (DBID) can be used as the OID to define a unique piece of data. In the SMARTS repository, it can be the instance name for a class (which is assumed to be unique).
5. For each object, retrieves other attributes as required. The attribute data can map to other columns in the tables for a relational database, or it can map to class attributes in the SMARTS repository.

When combining Rule Expressions, the Rule Evaluator must group them into queries specific to the needs of the data-storage resource. These queries can be:

- SQL queries, as in the case where the storage domain is a relational database management system.
- A custom query mechanism, as in the case where the storage domain is a SMARTS repository.

The Rule Evaluator depends on the Mapping Schema to format the query appropriately.

About the Mapping Schema

The Mapping Schema consists of a Parser and XML Schema file that specifies the mapping between groupable OGS classes and the elements in the application’s data-storage resource where the data for these objects is actually stored. The Rule Evaluator uses this information to convert rules into data-store-specific queries.

About the Rule Converter

The ASA infrastructure notation for a Rule Expression is different from the notation used by the OGS Server. This is necessary because the ASA can store extra information in the rule to help the Rule Evaluator generate queries.

Example

A Rule Expression for use with a SQL ASA can contain the entire query formed by the Query generator. The Rule Expression to be used with the Kilner VMSA can contain other types of information specific to a SMARTS Repository.

The Rule Converter converts any rule passed by the OGS Server into a rule format used by the Rule Validator and Rule Evaluator.

About Node Rule Expressions

A Node Rule Expression contains four elements:

1. The groupable class.
2. The groupable attribute.

CISCO CONFIDENTIAL

3. The groupable operator.
4. The value.

For example: In the Node Rule Expression `Device IPAddress contains "172.20"`, the class is `Device`, the attribute is `IPAddress`, the operator is `contains`, and the value is `"172.20"`.

Similar examples of Node Rule Expressions would include: `Device SystemLocation contains "US"`, or `Device SystemLocation contains "San Jose"`.

About Composite Rule Expressions

A Composite Rule Expression contains two or more Rule Expressions combined by the operators AND, OR or EXCLUDE.

The Rule Expressions combined in this way can be either Node Rule Expressions or other Composite Rule Expressions.

For example, we can create a Composite Rule Expression using just two Node Rule Expressions: `Device IPAddress contains "172.20" AND Device SystemLocation contains "US"`

We can also create a Composite Rule Expression that combines the foregoing Composite Rule Expression with another Node Rule Expression: `(Device IPAddress contains "172.20" AND Device SystemLocation contains "US") OR Device SystemLocation contains "San Jose"`.

About the ASA Change Alerter

The ASA Change Alerter informs the OGS Server whenever a change to an object takes place in the data-storage resource containing the object data. The ASA Change Alerter must inform OGS Server whenever:

- An object is added.
- An object is deleted.
- An object is modified.
- An individual object's attribute values changes.
- A groupable class's attribute values change..
- The domain data provider is available (that is, when the relational or object-oriented database comes up).

Customizing ASA Infrastructure Modules

The ASA infrastructure modules must be customized depending upon your application's requirements for:

- The types of objects your application will group.
- The way you want to group them.
- The way your application stores the object data.

The following topics provide guidelines and examples for customizing each of the ASA Infrastructure Modules:

- [Customizing the Rule Validator](#)
- [Customizing the Generic Schema](#)

CISCO CONFIDENTIAL

- [Customizing the Rule Evaluator](#)
- [Customizing the Mapping Schema](#)
- [Customizing the Rule Converter and Rule Expressions](#)
- [Customizing the ASA Change Alerter](#)

Customizing the Rule Validator

The Rule Validator function is provided by the default validator class `GenericValidatorImpl` in `com.cisco.nm.xml.ogs.asa`. You will rarely need to extend this class to customize it.

Customizing the Generic Schema

The Generic Schema parser has been designed for reuse with very little change. The Generic Schema parser function is provided by a default schema parser class, `OGSSchemaParser` in `com.cisco.nm.xml.ogs.server.parser`. You should reuse this class as needed.

It is not possible to create a default Generic Schema file that will work with any application. However, you can create a Generic Schema file for practically any combination of classes, attributes, operators and values using the following XML DTD:

```
<?xml version="1.0"?>
<!DOCTYPE domain [

  <!ELEMENT domain (application)>
  <!ATTLIST domain
    name CDATA #REQUIRED>

  <!ELEMENT application (class+,complexType+,group+)>
  <!ATTLIST application
    name CDATA #REQUIRED>

  <!ELEMENT class EMPTY>
  <!ATTLIST class
    name CDATA #REQUIRED
    type CDATA #REQUIRED
    groupable (yes|no) #REQUIRED >

  <!ELEMENT complexType (extension? , attribute*) >
  <!ATTLIST complexType
    name CDATA #REQUIRED>

  <!ELEMENT extension EMPTY>
  <!ATTLIST extension
    base CDATA #REQUIRED>

  <!ELEMENT attribute (operatorGroup?,restriction?) >
  <!ATTLIST attribute
    name CDATA #REQUIRED
    type CDATA #REQUIRED
    groupable (yes|no) #REQUIRED
    displayable (yes|no) #IMPLIED
    returnable (yes|no) #IMPLIED>

  <!ELEMENT operatorGroup EMPTY>
  <!ATTLIST operatorGroup
    ref CDATA #REQUIRED>
```

CISCO CONFIDENTIAL

```

<!ELEMENT restriction (enumeration*,minInclusive?,maxInclusive?) >

<!ELEMENT enumeration EMPTY>
<!ATTLIST enumeration
    value CDATA #REQUIRED
    alias CDATA #IMPLIED>

<!ELEMENT minInclusive EMPTY>
<!ATTLIST minInclusive
    value CDATA #REQUIRED>

<!ELEMENT maxInclusive EMPTY>
<!ATTLIST maxInclusive
    value CDATA #REQUIRED>

<!ELEMENT group (extension? , operator*) >
<!ATTLIST group
    name CDATA #REQUIRED>

<!ELEMENT operator EMPTY>
<!ATTLIST operator
    name CDATA #REQUIRED>

]>

<domain name="CMF">
  <application name="DCR">
    <class name = "Device" type="DeviceType" groupable="yes"/>

    <complexType name="DeviceType">
      <attribute name="DisplayName" type="string" groupable="yes"
displayable="yes" returnable="yes">
        <operatorGroup ref="StringOperatorsGroup"/>
      </attribute>

      <attribute name="ManagementIpAddress" type="string" groupable="yes"
displayable="no" returnable="yes">
        <operatorGroup ref="StringOperatorsGroup"/>
      </attribute>

      <attribute name="HostName" type="string" groupable="yes" displayable="no"
returnable="yes">
        <operatorGroup ref="StringOperatorsGroup"/>
      </attribute>

      <attribute name="DomainName" type="string" groupable="yes" displayable="no"
returnable="yes">
        <operatorGroup ref="StringOperatorsGroup"/>
      </attribute>

      <attribute name="DeviceIdentity" type="numeric" groupable="yes"
displayable="no" returnable="yes">
        <operatorGroup ref="EqualityOperatorsGroup"/>
      </attribute>

      <attribute name="SystemObjectID" type="string" groupable="yes"
displayable="no" returnable="yes">
        <operatorGroup ref="StringOperatorsGroup"/>
      </attribute>
    </complexType>
  </application>
</domain>

```

CISCO CONFIDENTIAL

```

    <attribute name="Category" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="Series" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="Model" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="MDFId" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="EqualityOperatorsGroup"/>
    </attribute>

    <attribute name="UDF0" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF1" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF2" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF3" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF4" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF5" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF6" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF7" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

    <attribute name="UDF8" type="string" groupable="yes" displayable="no"
returnable="yes">
    <operatorGroup ref="StringOperatorsGroup"/>
    </attribute>

```

CISCO CONFIDENTIAL

```

        <attribute name="UDF9" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

        <attribute name="UDF10" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

        <attribute name="UDF11" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

        <attribute name="UDF12" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

        <attribute name="UDF13" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

        <attribute name="UDF14" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

        <attribute name="UDF15" type="string" groupable="yes" displayable="no"
returnable="yes">
            <operatorGroup ref="StringOperatorsGroup"/>
        </attribute>

    </complexType>

    <group name="StringOperatorsGroup">
        <operator name="equals" />
        <operator name="contains" />
        <operator name="startswith" />
        <operator name="endswith" />
    </group>

    <group name="EqualityOperatorsGroup">
        <operator name="equals" />
    </group>

</application>
</domain>

```

Customizing the Rule Evaluator

The Rule Evaluator must implement the interface `QueryGeneratorIf` in class `com.cisco.nm.xms.ogs.asa`.

As the Rule Evaluator must query the application's data resource, no generic implementation of the Rule Evaluator exists. You must create a custom Rule Evaluator for your application and the data-storage resource with which your application works.

CISCO CONFIDENTIAL

However, implementations of the Rule Evaluator can be found at:

- `com.cisco.nm.xml.ogs.asa.genericsqlasa.SQLQueryGenerator`. This `SQLQueryGenerator` Rule Evaluator is designed to work with a SQL-compatible relational database management system, and with the custom Mapping Schema parser and file described in the “[Customizing the Mapping Schema](#)” section on page 30-27.
- `com.cisco.nm.xml.ogs.kilner10.vmsa.VMSAQueryGenerator`. This `VMSAQueryGenerator` Rule Evaluator is designed to work with the SMARTS Repository. It is a good example of how to work with object-oriented data sources.

`SQLQueryGenerator` uses the Generic SQL ASA module `SQLEvaluatorIf` to evaluate SQL queries in the database. The default version of the `SQLEvaluatorIf` interface is `CMFEvaluator` in `com.cisco.nm.xml.ogs.asa.genericsqlasa`. `CMFEvaluator` uses the CWCS database utility `DBService2` to evaluate SQL queries.

You must configure `CMFEvaluator` for your application by changing the “`Database_Name`” value in the `DBServer.properties` file to give the database name (e.g., “`RME`”) followed by “`_Implementation_Details.properties`”.

You need not provide the user name, password, data source URL, or other database properties. They will be picked up from `DBServer.properties` automatically.

Customizing the Mapping Schema

The Mapping Schema’s task of modeling the data source for the Rule Evaluator means that no default Mapping Schema implementation can exist, as the data-storage resource for each application can vary widely.

For example, the elements in the data resource can be:

- Tables, if the data source is a relational database.
- Classes, if the data source is an object-oriented data store.
- Files, if the data source relies on flat files.

Accordingly, you must write a Mapping Schema appropriate for the way your application sources and organizes its data. Since the Mapping Schema parser requires an XML file to parse, you must also create a Mapping Schema XML file customized for your application’s data structure.

The Generic SQL ASA is a customization of the ASA infrastructure for querying a SQL-compliant RDBMS. It uses a custom Mapping Schema parser (`RDBMSSchemaValidator` in class `com.cisco.nm.xml.ogs.asa.genericsqlasa`) which is in turn used by the `SQLQueryGenerator` Rule Evaluator.

`RDBMSSchemaValidator` was written to parse a Mapping Schema XML file conforming to the following DTD:

```
<?xml version="1.0"?>
<!DOCTYPE Domain [
<!ELEMENT Domain (Application)>
<!ATTLIST Domain
      Name CDATA #REQUIRED>

<!ELEMENT Application (Mappings)>
<!ATTLIST Application
      Name CDATA #REQUIRED>

<!ELEMENT Mappings (ClassMap+,group*)>
<!ATTLIST Application
```

CISCO CONFIDENTIAL

```

Name CDATA #REQUIRED>

<!ELEMENT ClassMap (OGSClass,ToTable,HierarchyInfo,PropertyMap+) >

<!ELEMENT OGSClass EMPTY>
<!ATTLIST OGSClass
    Name CDATA #REQUIRED>

<!ELEMENT ToTable (PrimaryKey) >
<!ATTLIST ToTable
    Name CDATA #REQUIRED>

<!ELEMENT PrimaryKey (Column+)>

<!ELEMENT Column (ColumnName,JDBC_Type,Operator)>
<!ELEMENT ColumnName (#PCDATA) >
<!ELEMENT JDBC_Type (#PCDATA) >
<!ELEMENT Operator (#PCDATA) >
<!ELEMENT HierarchyInfo (HierarchyType,ConditionalColumnInfo?) >
<!ELEMENT HierarchyType (#PCDATA) >
<!ELEMENT ConditionalColumnInfo (ColumnName,JDBC_Type,Operator,ColumnCondition) >
<!ELEMENT ColumnCondition (#PCDATA) >

<!ELEMENT PropertyMap (OGSAttribute,(AttributeToColumnMapping |
AttributeToTableDireCSTMapping | AttributeToTableIndireCSTMapping |
AttributeToTableConditionalDireCSTMapping),operatorMapGroup?)>

<!ELEMENT OGSAttribute EMPTY>
<!ATTLIST OGSAttribute
    Name CDATA #REQUIRED>

<!ELEMENT AttributeToColumnMapping (ColumnName,JDBC_Type)>

<!ELEMENT AttributeToTableDireCSTMapping (MappedTable,PrimaryToMappedAssociation)>
<!ELEMENT MappedTable EMPTY >
<!ATTLIST MappedTable
    Name CDATA #REQUIRED>

<!ELEMENT PrimaryToMappedAssociation (PrimaryTableColumnNames+,MappedTableColumnNames+)>

<!ELEMENT PrimaryTableColumnNames (PrimaryTableColumn+)>

<!ELEMENT PrimaryTableColumn (ColumnName) >

<!ELEMENT MappedTableColumnNames (MappedTableColumn+) >

<!ELEMENT MappedTableColumn (ColumnName) >

<!ELEMENT AttributeToTableConditionalDireCSTMapping
(MappedTable,PrimaryToMappedAssociation,MappedConditionalColumnName,MappedConditionalColumn
nValue,MappedValueColumnName)>

<!ELEMENT MappedConditionalColumnName (#PCDATA) >
<!ELEMENT MappedConditionalColumnValue (#PCDATA) >
<!ELEMENT MappedValueColumnName (#PCDATA) >

<!ELEMENT AttributeToTableIndireCSTMapping
(MappedTable,IndirectTable,IndirectToMappedAssociation,IndirectToPrimaryAssociation) >
<!ELEMENT IndirectTable EMPTY >
<!ATTLIST IndirectTable
    Name CDATA #REQUIRED>

```

CISCO CONFIDENTIAL

```

<!ELEMENT IndirectToMappedAssociation (IndirectTableColumnNames+,MappedTableColumnNames+)
>
<!ELEMENT IndirectTableColumnNames (IndirectTableColumn+) >

<!ELEMENT IndirectTableColumn (ColumnName) >

<!ELEMENT IndirectToPrimaryAssociation
(IndirectTableColumnNames+,PrimaryTableColumnNames+) >

<!ELEMENT operatorMapGroup EMPTY>
<!ATTLIST operatorMapGroup
    ref CDATA #REQUIRED>

<!ELEMENT group (OperatorMap+) >
<!ATTLIST group
    Name CDATA #REQUIRED>

<!ELEMENT OperatorMap (OGSOperator,ToOperator) >

<!ELEMENT OGSOperator EMPTY>
<!ATTLIST OGSOperator
    Name CDATA #REQUIRED>

<!ELEMENT ToOperator (SQLOperator) >

<!ELEMENT SQLOperator EMPTY>
<!ATTLIST SQLOperator
    Name CDATA #REQUIRED>

]>

```

The RDBMSSchemaValidator DTD was written to allow you to map data from any SQL-compliant relational DBMS. It lets you specify exactly how individual OGS classes map to RDBMS tables and how to map OGS attributes, operators and values to table columns.

With it, you can create Mapping Schema XML files that will handle all aspects of the task of mapping OGS classes to RDBMS tables.

The following examples demonstrate how to do this for many of the most typical cases.

Example: One Table Maps to One Concrete Class

In this case, the RDBMS contains a table called “DeviceTable”. It has many columns, including “SysName” and “SysLocation”. Using the RDBMSSchemaValidator DTD, we can specify that the OGS class “Device” maps directly to “DeviceTable”.

Similarly, the OGS class “Device” can have the attributes “SysName” and “SysLocation”. These OGS attributes map directly to the corresponding columns in “DeviceTable”.

In the Mapping Schema XML file, you can make this type of mapping explicit by supplying the value `ONE_TABLE_PER_CONCRETE_CLASS` for the element *HierarchyType* inside the element *HierarchyInfo*.

**Note**

The Generic SQL ASA Rule Evaluator (SQLQueryGenerator) currently supports conversion of rules into SQL queries for Hierarchy classes of the type given in this example. You need not modify SQLQueryGenerator for this purpose.

CISCO CONFIDENTIAL**Example: One Table Maps to a Hierarchy of Classes**

In this example, the RDBMS table called “Device Table” covers all device types. Each row in the table can contain data for a different type of device, such as “Router”, “Switch”, or “PhoneAccessSwitch”. Every row has the column “DeviceType”, which has a different value depending on the type of device described in that row.

Using the RDBMSSchemaValidator DTD, we can design many OGS classes for this mapping. One of the simplest would be to define a base OGS class called “Device”, with many subclasses for “Router”, “Switch”, etc.

All of these subclasses can also map to “DeviceTable”, but with a condition on the “DeviceType” column. In the XML file, the <HierarchyType> for these classes will be ONE_TABLE_PER_HIERARCHY.



Note The Generic SQL ASA Rule Evaluator (SQLQueryGenerator) can *not* convert rules into SQL queries for this Hierarchy class. If you want to create table mappings for this format, you should implement your own SQLQueryGenerator in the lines of the default implementation.

Example: Multiple Tables Map to One Abstract Class

In this case, the RDBMS has no single “Device” table. Instead, it has a table for each kind of device: “Router”, “Switch”, “PhoneAccessSwitch”, and so on.

To handle this, we create an abstract base OGS class called “Device” which does not map to any DB table, and concrete subclasses of “Device”, like “Router” and “Switch”, each of which maps to the corresponding RDBMS table.

The objects in the “Device” class are just a summation of the objects in the concrete subclasses. The set of OGS attributes common to each of the concrete subclasses belongs to the parent “Device” OGS class, so the similarly named columns in the tables for individual devices are integrated into the class hierarchy. In the Mapping Schema XML file, the *HierarchyType* for these classes is ONE_TABLE_PER_CLASS.



Note The Generic SQL ASA Rule Evaluator (SQLQueryGenerator) can *not* convert rules into SQL queries for this Hierarchy class. If you want to create table mappings for this format, you should implement your own SQLQueryGenerator in the lines of the default implementation.

Example: Mapping Class Attributes to Table Columns

We have the OGS class “Device” mapped to the RDBMS table “Device”, and the OGS “Device” attribute “SystemLocation” mapped to the column “SystemLocation” in the “Device” table. We could write this in the Mapping Schema XML file as:

```
<Domain Name="SomeDomainName">
<Application Name="OGS">
<Mappings>
<ClassMap>
  <OGSClass Name="Device"/>
  <ToTable Name="Device">
    <PrimaryKey>
      <Column>
        <ColumnName>DeviceID</ColumnName>
        <JDBC_Type>INTEGER</JDBC_Type>
        <Operator>=</Operator>
      </Column>
    </PrimaryKey>
  </ToTable>
</HierarchyInfo>
```


CISCO CONFIDENTIAL

```

<HierarchyType>ONE_TABLE_PER_CONCRETE_CLASS</HierarchyType>
</HierarchyInfo>

<PropertyMap>
  <OGSAttribute Name="SystemLocation"/>
  <AttributeToColumnMapping>
    <ColumnName>SystemLocation</ColumnName>
    <JDBC_Type>VARCHAR</JDBC_Type>
  </AttributeToColumnMapping>
  <operatorMapGroup ref="StringOperatorMap"/>
</PropertyMap>
</ClassMap>

<group Name="StringOperatorMap">

  <OperatorMap>
    <OGSOperator Name="equals"/>
    <ToOperator>
      <SQLOperator Name="="/>
    </ToOperator>
  </OperatorMap>

  <OperatorMap>
    <OGSOperator Name="contains"/>
    <ToOperator>
      <SQLOperator Name="LIKE"/>
    </ToOperator>
  </OperatorMap>
</group>

</Mappings>
</Application>
</Domain>

```

In this example, the column “SystemLocation” is of type “VARCHAR”. The mappings for the OGS operators for this OGS attribute type, “equals” and “contains”, are defined in the operators group, “StringOperatorMap”.

**Note**

The Generic SQL ASA Rule Evaluator (SQLQueryGenerator) currently supports the kind of attribute mapping shown in this example

Examples: Mapping Attributes Across Tables Using Foreign Keys

If you want to have an OGS class called “Device” with the attributes “IPAddress” and “IPSubnetMask”. But the RDBMS tables have:

- Table “Device”, with columns “SystemLocation”, “SysObjectID”, “SystemContact” and “DeviceID”.
- Table “IPTable”, with columns “DevID”, “IPAddress” and “IPSubnetMask”. “DevID” is a foreign key pointing to the primary key “DeviceID” on table “Device”.

You could design the classes and mapping as follows:

- OGS class “Device” maps to table “Device”.
- Class “Device” has the attributes “SystemLocation”, “SystemContact” and “SysObjectID”, which map to columns with the same names in table “Device”.
- Class “Device” contains a “composite” attribute of type "Class IP".

CISCO CONFIDENTIAL

- Class “IP” maps to table “IPTable”.
- Class “IP” has the attributes “IPAddress” and “IPSubnetMask” mapped to the columns with the same names in table “IPTable”.

The Mapping Schema XML file for this design would contain:

```
<Domain Name="SomeDomainName">
<Application Name="OGS">
<Mappings>
<ClassMap>
  <OGSClass Name="Device" />
  <ToTable Name="Device">
    <PrimaryKey>
      <Column>
        <ColumnName>DeviceID</ColumnName>
        <JDBC_Type>INTEGER</JDBC_Type>
        <Operator>=</Operator>
      </Column>
    </PrimaryKey>
  </ToTable>

  <HierarchyInfo>
  <HierarchyType>ONE_TABLE_PER_CONCRETE_CLASS</HierarchyType>
  </HierarchyInfo>

  <PropertyMap>
    <OGSAttribute Name="IP" />
    <AttributeToTableDireCSTMapping>
      <MappedTable Name="IPTable" />
      <PrimaryToMappedAssociation>
        <PrimaryTableColumnNames>
          <PrimaryTableColumn>
            <ColumnName>DeviceID</ColumnName>
          </PrimaryTableColumn>
        </PrimaryTableColumnNames>
        <MappedTableColumnNames>
          <MappedTableColumn>
            <ColumnName>DevID</ColumnName>
          </MappedTableColumn>
        </MappedTableColumnNames>
      </PrimaryToMappedAssociation>
    </AttributeToTableDireCSTMapping>
  </PropertyMap>
</ClassMap>

<ClassMap>
  <OGSClass Name="IP" />
  <ToTable Name="IPTable">
    <PrimaryKey>
      <Column>
        <ColumnName>DevID</ColumnName>
        <JDBC_Type>INTEGER</JDBC_Type>
        <Operator>=</Operator>
      </Column>
      <Column>
        <ColumnName>IPAddress</ColumnName>
        <JDBC_Type>VARCHAR</JDBC_Type>
        <Operator>=</Operator>
      </Column>
    </PrimaryKey>
  </ToTable>

  <HierarchyInfo>
```

CISCO CONFIDENTIAL

```

<HierarchyType>ONE_TABLE_PER_CONCRETE_CLASS</HierarchyType>
</HierarchyInfo>

<PropertyMap>
  <OGSAttribute Name="IPAddress" />
  <AttributeToColumnMapping>
    <ColumnName>IPAddress</ColumnName>
    <JDBC_Type>VARCHAR</JDBC_Type>
  </AttributeToColumnMapping>
  <operatorMapGroup ref="StringOperatorMap" />
</PropertyMap>

<PropertyMap>
  <OGSAttribute Name="IPSubnetMask" />
  <AttributeToColumnMapping>
    <ColumnName>IPSubnetMask</ColumnName>
    <JDBC_Type>VARCHAR</JDBC_Type>
  </AttributeToColumnMapping>
  <operatorMapGroup ref="StringOperatorMap" />
</PropertyMap>

</ClassMap>

<group Name="StringOperatorMap">

  <OperatorMap>
    <OGSOperator Name="equals" />
    <ToOperator>
      <SQLOperator Name="=" />
    </ToOperator>
  </OperatorMap>

  <OperatorMap>
    <OGSOperator Name="contains" />
    <ToOperator>
      <SQLOperator Name="LIKE" />
    </ToOperator>
  </OperatorMap>
</group>

</Mappings>
</Application>
</Domain>

```

**Note**

The Generic SQL ASA Rule Evaluator (SQLQueryGenerator) currently supports the kind of attribute mapping shown in this example

Example: Mapping Attributes Across Multiple Tables

You want to have an OGS class “ICS” to represent a chassis containing multiple devices. “ICS” must have an attribute “IPAddress” for each one of its contained SPE devices, so that we can sort on it.

The RDBMS tables are as follows:

- Table “ICS” contains columns with all relevant ICS chassis data.
- Column “ICSID” is the primary key of table “ICS”.
- Table “Device” has columns for all relevant device information, including:
 - An “ICSID” column, which is a foreign key pointing to the primary key of table “ICS”.
 - A “DeviceID” column, which uniquely identifies each SPE device in the chassis.

CISCO CONFIDENTIAL

- Table “IP” contains:
 - “IPAddress” and “Subnet” columns for the various interfaces for each device within a chassis.
 - A “DevID” column, which is a foreign key pointing to the primary key “DeviceID” of table “Device”.

You can design a solution as follows:

- Class “ICS” maps to table “ICS”.
- Class “ICS” has one attribute, “IPAddress”, which is of type "IP Class".
- Attribute “IPAddress” maps to table “IP”. The attribute “IP” contains information on the PK,FK relationship between the tables “ICS”, “Device” and “IP”.

The Mapping Schema XML file for this design is as follows:

```
<Domain Name="SomeDomainName">
<Application Name="OGS">
<Mappings>
<ClassMap>
  <OGSClass Name="Device"/>
  <ToTable Name="Device">
    <PrimaryKey>
      <Column>
        <ColumnName>DeviceID</ColumnName>
        <JDBC_Type>INTEGER</JDBC_Type>
        <Operator>=</Operator>
      </Column>
    </PrimaryKey>
  </ToTable>

  <HierarchyInfo>
  <HierarchyType>ONE_TABLE_PER_CONCRETE_CLASS</HierarchyType>
  </HierarchyInfo>

<PropertyMap>
  <OGSAttribute Name="IP"/>
  <AttributeToTableIndireCSTMapping>
    <MappedTable Name="IP" />
    <IndirectTable Name="Device" />

    <IndirectToMappedAssociation>
      <IndirectTableColumnNames>
        <IndirectTableColumn>
          <ColumnName>DeviceID</ColumnName>
        </IndirectTableColumn>
      </IndirectTableColumnNames>
      <MappedTableColumnNames>
        <MappedTableColumn>
          <ColumnName>DevID</ColumnName>
        </MappedTableColumn>
      </MappedTableColumnNames>
    </IndirectToMappedAssociation>

    <IndirectToPrimaryAssociation>

      <IndirectTableColumnNames>
        <IndirectTableColumn>
          <ColumnName>ICSID</ColumnName>
        </IndirectTableColumn>
      </IndirectTableColumnNames>
      <PrimaryTableColumnNames>
        <PrimaryTableColumn>
          <ColumnName>ICSID</ColumnName>
```

CISCO CONFIDENTIAL

```

        </PrimaryTableColumn>
    </PrimaryTableColumnNames>

    </IndirectToPrimaryAssociation>
</AttributeToTableIndireCSTMapping>
</PropertyMap>

</ClassMap>

<ClassMap>
  <OGSClass Name="IP" />
  <ToTable Name="IPTable">
    <PrimaryKey>
      <Column>
        <ColumnName>DevID</ColumnName>
        <JDBC_Type>INTEGER</JDBC_Type>
        <Operator>=</Operator>
      </Column>
      <Column>
        <ColumnName>IPAddress</ColumnName>
        <JDBC_Type>VARCHAR</JDBC_Type>
        <Operator>=</Operator>
      </Column>
    </PrimaryKey>
  </ToTable>

  <HierarchyInfo>
  <HierarchyType>ONE_TABLE_PER_CONCRETE_CLASS</HierarchyType>
  </HierarchyInfo>

  <PropertyMap>
    <OGSAttribute Name="IPAddress" />
    <AttributeToColumnMapping>
      <ColumnName>IPAddress</ColumnName>
      <JDBC_Type>VARCHAR</JDBC_Type>
    </AttributeToColumnMapping>
    <operatorMapGroup ref="StringOperatorMap" />
  </PropertyMap>

  <PropertyMap>
    <OGSAttribute Name="IPSubnetMask" />
    <AttributeToColumnMapping>
      <ColumnName>IPSubnetMask</ColumnName>
      <JDBC_Type>VARCHAR</JDBC_Type>
    </AttributeToColumnMapping>
    <operatorMapGroup ref="StringOperatorMap" />
  </PropertyMap>

</ClassMap>

<group Name="StringOperatorMap">

  <OperatorMap>
    <OGSOperator Name="equals" />
    <ToOperator>
      <SQLOperator Name="=" />
    </ToOperator>
  </OperatorMap>

  <OperatorMap>
    <OGSOperator Name="contains" />
    <ToOperator>
      <SQLOperator Name="LIKE" />
    </ToOperator>
  </OperatorMap>
</group>

```

CISCO CONFIDENTIAL

```
</OperatorMap>
</group>

</Mappings>
</Application>
</Domain>
```

**Note**

The Generic SQL ASA Rule Evaluator (SQLQueryGenerator) currently supports the kind of attribute mapping shown in this example

Customizing the Rule Converter and Rule Expressions

The default implementation of the Rule Converter is RuleConverterImpl in com.cisco.nm.xmls.ogs.asa. This Rule Converter is re-used by most applications.

The Node Rule Expression is modeled by class NodeRuleExpression in com.cisco.nm.xmls.ogs.asa. The class CompositeRuleExpression in com.cisco.nm.xmls.ogs.asa models the Composite Rule Expression. Extensions of NodeRuleExpression and CompositeRuleExpression created to hold ASA-specific information must be maintained in product-specific directories.

For examples of custom Rule Expressions, see the following Generic SQL ASA implementations:

- SQLNodeRuleExpression in com.cisco.nm.xmls.ogs.asa.genericsqlasa.
- SQLCompositeRuleExpression in com.cisco.nm.xmls.ogs.asa.genericsqlasa.

Customizing the ASA Change Alerter

ASA Change Alerters are rarely customized. The interface that defines the ASA Change Alerter functions is ASACHangeAlertIf in the class com.cisco.nm.xmls.ogs.asa. The default implementation is ASACHangeAlerterBase, also in com.cisco.nm.xmls.ogs.asa.

Running a Customized ASA

After you have created your application ASA, you must implement it, to ensure that the OGS Server will instantiate it at runtime.

The following topics explain the tasks necessary to run your customized ASA:

- [Registering the ASA with OGS](#)
- [Creating the ASA Configuration File](#)
- [Example: Using the Generic SQL ASA](#)

Registering the ASA with OGS

To register an ASA with the OGS Server, you must create a registration file for the ASA and specify it in the OGSServer.properties file.

The OGS Server uses the ASA registration file to:

- Instantiate an ASA at startup.
- Assign to this ASA instance the name specified in the registration file.

CISCO CONFIDENTIAL

The ASA registration file's name must appear in the OGSServer.properties file as the value of the `ASARegistrationFile` parameter. You are free to assign this file any unique name, although it is customary to name it as follows:

```
ASARegistrationFile=XXXX-mapping.xml
```

where `xxx` is the application name.

The only real restrictions on the ASA registration file are that its filename must end in the extension “xml” and that it must conform to the following DTD:

```
<?xml version="1.0"?>
<!DOCTYPE ASA_Mappings [
<!ELEMENT ASA_Mappings (Mapping+) >

<!ELEMENT Mapping (OGSSchemaFile+,Location+,ClassName+,Specialization+)>

<!ELEMENT OGSSchemaFile EMPTY>
<!ATTLIST OGSSchemaFile
    name CDATA #REQUIRED >

<!ELEMENT Location EMPTY>
<!ATTLIST Location
    type (Remote|Local) #REQUIRED >

<!ELEMENT ClassName EMPTY>
<!ATTLIST ClassName
    name CDATA #REQUIRED >

<!ELEMENT Specialization EMPTY>
<!ATTLIST Specialization
    name CDATA #REQUIRED >

]>
```

Example

You have created an ASA for one of the applications in Kilner, called “VMSA”. You have created an ASA Registration File, called `vmsa-mapping.xml`, for the VMSA ASA. It has the following content:

```
<?xml version="1.0"?>
<!DOCTYPE ASA_Mappings [
<!ELEMENT ASA_Mappings (Mapping+) >

<!ELEMENT Mapping (OGSSchemaFile+,Location+,ClassName+,Specialization+)>

<!ELEMENT OGSSchemaFile EMPTY>
<!ATTLIST OGSSchemaFile
    name CDATA #REQUIRED >

<!ELEMENT Location EMPTY>
<!ATTLIST Location
    type (Remote|Local) #REQUIRED >

<!ELEMENT ClassName EMPTY>
<!ATTLIST ClassName
    name CDATA #REQUIRED >

<!ELEMENT Specialization EMPTY>
<!ATTLIST Specialization
    name CDATA #REQUIRED >

]>
```

CISCO CONFIDENTIAL

```

<ASA_Mappings>
  <Mapping>
    <OGSSchemaFile name="vms-ogs-schema.xml" />
    <Location type="Local" />
    <ClassName name="com.cisco.nm.xml.ogs.asa.ASABaseImpl" />
    <Specialization name="VMSA" />
  </Mapping>
</ASA_Mappings>

```

When creating an ASA Registration file, note that:

- You must either include the DTD in the file or reference an external location for the DTD.
- Although you can enter either “local” or “remote” as the “Location type” value, this version of OGS will process only “local”.
- You must specify a Specialization name if you are using ASABaseImpl. OGS will use the Specialization name as a tag to locate the property files and modules used by this instance.

Creating the ASA Configuration File

The ASA configuration file tells a newly instantiated ASA which infrastructure modules to use when validating and evaluating rules. It acts, essentially, as a list of all the customizations performed on the generic ASA implementation modules.

The ASA configuration filename must always:

- Start with the name of your application as you registered it with the OGS Server in the OGSServer.properties file.
- End with the string “_Implementation_Details.properties”.

Example

You have created an ASA for one of the applications in Kilner, called “VMSA”. The ASA configuration file for this application will be named “VMSA_Implementation_Details.properties”.

The ASA configuration file content must list:

- All of the ASA infrastructure modules the ASA uses, whether generic or customized. In the VMSA ASA example, we have created custom versions or re-used generic versions of the components shown in [Table 30-3](#).
- The Generic Schema filename. In the VMSA ASA, this file is named “vms-ogs-schema.xml”.
- The Mapping Schema filename. In the VMSA ASA, this file is “vmsa-mapping-schema.xml”.

Table 30-3 VMSA ASA Infrastructure Modules

Component	Class	In	Type
Rule Validator	GenericValidatorImpl	com.cisco.nm.xml.ogs.asa.	Generic
Generic Schema Parser	OGSSchemaParser	com.cisco.nm.xml.ogs.server.parser	Generic
Rule Evaluator	VMSAQueryGenerator	com.cisco.nm.xml.ogs.kilner10.vmsa	Custom
Mapping Schema Parser	VMSAValidator	com.cisco.nm.xml.ogs.kilner10.vmsa	Custom
Rule Converter	RuleConverterImpl	com.cisco.nm.xml.ogs.asa	Generic
Node Rule Expression	VMSANodeRuleExpression	com.cisco.nm.xml.ogs.kilner10.vmsa	Custom

CISCO CONFIDENTIAL**Table 30-3 VMSA ASA Infrastructure Modules (continued)**

Component	Class	In	Type
Composite Rule Expression	VMSACompositeRuleExpression	com.cisco.nm.xml.ogs.kilner10.vmsa	Custom
ASA Change Alerter	VmsaChangeAlerter	com.cisco.nm.xml.ogs.kilner10.vmsa	Custom

The ASA Configuration File lists this information using the parameters shown in [Table 30-4](#). The Configuration File is a simple ASCII text file containing name=value pairs. The order in which the parameters appear in the file is not important.

However, you must:

- Use all of the parameter names shown in [Table 30-4](#), exactly as they appear in the table.
- Supply complete values for each parameter.

Table 30-4 ASA Configuration File Parameters

For this component	Use this parameter	And assign to it the
Rule Validator	Rule_Validator=	Classname of your Rule Validator
Generic Schema Parser	Schema_Validator_Class=	Classname of your Generic Schema Parser
Generic Schema File	Generic_Schema_File=	Path and name of the XML schema file used by the Generic Schema Parser.
Rule Evaluator	Rule_Evaluator=	Classname of your Rule Evaluator
Mapping Schema Parser	Class_Mapping_Parser=	Classname of your Mapping Schema Parser.
Mapping Schema File	Mapping_File=	The path and name of the XML schema file used by the Mapping Schema Parser.
Rule Converter	Rule_Converter=	Classname of your Rule Converter
Node Rule Expression	Node_Expression_Class=	Classname of your Node Rule Expression model.
Composite Rule Expression	Composite_Expression_Class=	Classname of your Composite Rule Expression model.
ASA Change Alerter	Asa_Change_Alerter=	Classname of your ASA Change Alerter.

The VMSA ASA Configuration File might look like this:

```
Rule_Validator = com.cisco.nm.xml.ogs.asa.GenericValidatorImpl
Schema_Validator_Class = com.cisco.nm.xml.ogs.server.parser.OGSSchemaParser
Generic_Schema_File = /vms-ogs-schema.xml
Rule_Evaluator = com.cisco.nm.xml.ogs.kilner10.vmsa.VMSAQueryGenerator
Class_Mapping_Parser = com.cisco.nm.xml.ogs.kilner10.vmsa.VMSAValidator
Mapping_File = /vms-mapping-schema.xml
Rule_Converter = com.cisco.nm.xml.ogs.asa.RuleConverterImpl
Node_Expression_Class = com.cisco.nm.xml.ogs.kilner10.vmsa.VMSANodeRuleExpression
Composite_Expression_Class =
com.cisco.nm.xml.ogs.kilner10.vmsa.VMSACompositeRuleExpression
Asa_Change_Alerter = com.cisco.nm.xml.ogs.kilner10.vmsa.VmsaChangeAlerter
```

CISCO CONFIDENTIAL**Example: Using the Generic SQL ASA**

The Generic SQL ASA is a customization of the ASA infrastructure for querying a relational database using OGS. It uses the generic or customized components shown in [Table 30-5](#).

Table 30-5 Generic SQL ASA Components

Component	Class	Status
Rule Validator	GenericValidatorImp in com.cisco.nm.xmls.ogs.asa	Generic
Generic Schema Parser	OGSSchemaParser in com.cisco.nm.xmls.ogs.server.parser.	Generic
Rule Evaluator	SQLQueryGenerator in com.cisco.nm.xmls.ogs.asa.genericsqlasa.	Custom
Rule Converter	RuleConverterImpl in com.cisco.nm.xmls.ogs.asa.	Generic
Mapping Parser	RDBMSSchemaValidator in com.cisco.nm.xmls.ogs.asa.genericsqlasa.	Custom
Node Expression	SQLNodeRuleExpression in com.cisco.nm.xmls.ogs.asa.genericsqlasa.	Custom
Composite Expression	SQLCompositeRuleExpression in com.cisco.nm.xmls.ogs.asa.genericsqlasa.	Custom

Creating an OGS GUI

The UII (User Interface Initiative) Object Selector provides a convenient and versatile tree-display component that gives users GUI access to OGS group data. When implemented, it provides users with:

- A group hierarchy display.
- A means of selecting objects within groups, including both single and multiple selection.

Object Selector is part of the UII release package available at the User Experience web site, which is available to Cisco employees at <http://picasso> (be sure to use version 6.1 or later of the UII release package).

Object Selector comes in standard “Content Area” and space-conserving “Sliding” versions, and has a complete set of API functions. If you plan on using Object Selector in your application, please remember:

- Integration between Object Selector and OGS is application-specific. You should refer to the *SDK Developer’s Guide for UII* (the SDK for UII Release 6.1 is available as EDCS-348564) or the online help version of the same document at <http://picasso>, for details on programming Object Selector to work with your application.
- As part of your integration effort, you will need to develop an Object Selector Tree Generator which retrieves data from OGSServerProxy and converts it for display in Object Selector. This Tree Generator is part of the application deliverable, and must be maintained by you.
- OGS supplies a default Object Selector Tree Generator for use by the OGS Administration GUI. This default Tree Generator can serve as a model for a custom Tree Generator for your application:
 - The default Tree generator is ObjectSelectorTreeGenerator in com.cisco.nm.xmls.client.ostaglib.util.

CISCO CONFIDENTIAL

- For source code, set up an ogs1_1 view in ClearCase, then view:
/vob/enm_ogs/share/classes/client/com/cisco/nm/xms/ogs/client/ostaglib/util/ObjectSelectorTreeGenerator.java.
- If you plan on using Object Selector with an OGS implementation that includes Secure Views, see the “Using Secure Views with Object Selector” section on page 30-46

To use the Object Selector in your application:

-
- Step 1** Install the latest UII package following the instructions in the [UII SDK](#).
- Step 2** Copy the following directories and files from the UII package to the corresponding directories of your project, under the Tomcat webapps/YOUR_APPLICATION directory.
- /objectselector
 - /WEB-INF/lib/ogs-objselector.jar
 - /WEB-INF/tlds/ogs-objectselector-taglib.tld
- Step 3** Insert the following lines into the *action-mappings* section of your project's /WEB-INF/struts-config.xml file.
- ```
<action path="/objselcaching"
type="com.cisco.nm.xms.ogs.client.uii.ObjectSelectorCachingAction"/>
<action path="/objselsliding"
type="com.cisco.nm.xms.ogs.client.uii.ObjectSelectorSlidingPanelAction"/>
```
- Step 4** Using the Dreamweaver Extension manager, install the included Dreamweaver Extension file, OGS-OS-DWX.mxp.
- Step 5** Drag and drop into your JSP file the icon corresponding to the type of Object Selector you want to create. In order to preserve the selection, put it into the current forms[0] (within the first <uii:form> tag of the page).

If you are not using Dreamweaver, insert directly into the corresponding JSP file the code appropriate for the type of Object Selector you want to create:

- For a Sliding Object Selector, insert this code:

```
<%@ taglib uri="/WEB-INF/tlds/ogs-objectselector-taglib.tld" prefix="ogs" %>
<ogs:slidingSelector
osName="YOUR_OBJECT_SELECTOR_NAME"
selectionMode="multiple"
preserveSelection="false"
isGroupSelector="false"
mixSelection="false"
showFilter="true"
showQuickFilter="true"
showSaveBtn="true"
treeGenerator="com.cisco.nm.xms.ogs.test.TestTreeGenerator"/>
```

- For a Content Area Object Selector, insert this code:

```
<%@ taglib uri="/WEB-INF/tlds/ogs-objectselector-taglib.tld" prefix="ogs" %>
<ogs:contentAreaSelector
osName="YOUR_OBJECT_SELECTOR_NAME"
selectionMode="multiple"
preserveSelection="false"
isGroupSelector="false"
mixSelection="false"
width="200"
height="300"
treeGenerator="com.cisco.nm.xms.ogs.test.TestTreeGenerator"/>
```

**CISCO CONFIDENTIAL**

- Once you have inserted the code, set the attributes in the JSP tags to match your requirements.

**Step 6** Implement the JavaScript functions as needed. For details on the available functions, see the [UII SDK](#).

**Step 7** In the corresponding Action class, which handles the form containing the object selector, instantiate as the first object in the “perform” method an instance of the class `OGSObjectSelectorAdapter` (in `com.cisco.nm.xml.ogs.client.uii`).

**Step 8** Implement a custom Tree Generator class, as follows:

- Create a class extended from `BasicObjectSelectorTreeGenerator` in `com.cisco.nm.xml.ogs.client.ostaglib.util`.
- Set this class name in the JSP tag attribute `treeGenerator`.

If you do not do this, the Object Selector will call the default tree generator class, `ObjectSelectorTreeGenerator` in `com.cisco.nm.xml.ogs.client.ostaglib.util`. This class also extends from `BasicObjectSelectorTreeGenerator` and is provided by the application.

For reference information on the Tree Generator, see the [UII SDK](#).

- In this tree generator class, implement the following methods:

```

getTreeEntries REQUIRED
getUserFilterEntries OPTIONAL (For sliding object selector only).
getAppletParameterImages OPTIONAL
getAppletParameterImageFile OPTIONAL
getAppletParameterToolTipFile OPTIONAL
getAppletParameterToolTipLineContinue OPTIONAL
getAppletParameterToolTipWidth OPTIONAL
getAppletParameterBkgColor OPTIONAL
getAppletParameterBkgColorSelect OPTIONAL
getAppletParameterBkgColorHover OPTIONAL
getAppletParameterTabOpenColor OPTIONAL
getAppletParameterTabCloseColor OPTIONAL
getAppletParameterFontName OPTIONAL
getAppletParameterFontSize OPTIONAL
getAppletParameterPadding OPTIONAL
getAppletParameterOnSelect OPTIONAL
getAppletParameterOnUnSelect OPTIONAL
getAppletParameterOnCachingExpand OPTIONAL
getAppletParameterSelectAllChild OPTIONAL
getAppletParameterUnSelectAllChild OPTIONAL
getAppletParameterSelectAncestors OPTIONAL
getAppletParameterUnSelectAncestors OPTIONAL
getAppletParameterTabs OPTIONAL
getAppletParameterCheckWhenLabelClick OPTIONAL
getAppletParameterOnHighLight OPTIONAL
getAppletParameterDisable OPTIONAL
getAppletParameterSort OPTIONAL
getAppletParameterXml OPTIONAL
getAppletParameterXmlSource OPTIONAL
getAppletParameterDeDuplicate OPTIONAL
getAppletParameterNumberOfSelectedLeaves OPTIONAL

```

For more information about these methods, see the [UII SDK](#).

**CISCO CONFIDENTIAL**

## Using OGS Secure Views

Starting with CWCS 3.0, OGS supports Secure Views. OGS Secure Views allow your application to control user access to individual members of a group, instead of just to groups as a whole.

Using the user ID, application ID, and task ID (and optionally, the session ID) of the requesting client, and an interface to the CAM-maintained ACS Server security system, Secure Views can return a subset of the total membership of a group. This subset will contain only those devices to which the user has access.

OGS Secure Views are explained in the following topics:

- [How Secure Views Work](#)
- [Implementing Secure Views](#)
- [Customizing Your Secure Views Implementation](#)

## How Secure Views Work

OGS Secure Views filter the memberships of a group based on the IDs of:

- The user requesting access to a group's members.
- The application the user is using to access the group's members.
- The task the user intends to perform on the group's members.

The basic process flow proceeds as follows:

1. The OGSCient gets a complete list of objects for the requested group from the OGS Server.
2. The OGSCient gets from CAM the list of devices that the user is authorized to use for the given task, application and (optionally) session ID.
3. The OGSCient performs an intersection of the two lists:
  - a. It ignores non-device objects, passing them through to the OGSOBJECTLIST.
  - b. It filters out of the OGSOBJECTLIST all device objects that do not also exist in the CAM list.
  - c. The OGSCient returns to the application GUI an OGSOBJECTLIST containing all non-device objects and only filtered device objects.

OGS Secure Views accomplishes this using the following classes and methods new in CWCS 3.0:

- The basic interface class OGSFilterIf, in com.cisco.nm.xml.ogs.client.

The default implementation is OGSFilterImpl in the same package. OGSFilterImpl looks up the value of the "DeviceClasses" property in OGSCient.properties, uses it to identify all of the device classes that need to be filtered, and checks in the CAM-returned list for objects of those classes.

Only when the class of object in the CAM list matches the "DeviceClasses" value will the object be filtered. Applications using SQLASA need to use the OGSSQLFilterImpl implementation.

- OGSManagementFormAction class in com.cisco.nm.xml.ogs.client.mgmt
- OGSServerProxy in com.cisco.nm.xml.ogs.client. OGSServerProxy detects if the client requesting group or rule evaluation is an OGSAdminClient. It does this by looking for the Application ID "OGS" and Task ID "OGSOPERATION" in OGSSecurityContext.

For details, see the ["Using Secure Views With the OGS Administrative GUI" section on page 30-47](#)).

**CISCO CONFIDENTIAL**

- Class `OGSSecurityContext`, which defines a constructor that accepts the user ID, application ID, task ID and session ID (which can be null), that CAM requires to perform device filtering.

For details, see the [“Using OGS SecurityContext” section on page 30-45](#)

The design of OGS Secure Views imposes several requirements on applications using it:

- While all the filtering of the membership list is actually performed on the client, Secure Views depends entirely on the CWCS CAM security interface to get the list of group members for which the user is authorized.

It therefore requires that your application operate in ACS (TACACS+) mode, not in CMF security mode. If your application operates in CMF mode, the default implementation of Secure Views will not perform any filtering at all. In this case, evaluating a group or rule will result in all members of that group being returned.

- You must always specify the OGS classes your application uses to represent devices. You do this using the `"DeviceClasses"` property in the OGS client's `OGSClient.properties` file.

For example: An `OGSClient.properties` entry of `DeviceClasses="CMF:DCR:Device"` specifies that objects of the class `"CMF:DCR:Device"`, and only this class, represent devices. Similarly, the entry `DeviceClasses="CMF:DCR:Device","CMF:DCR:Interface"` specifies that both these classes represent devices.

If you do not do this, your Secure Views implementation will pass all device objects through to the client, regardless of the user's authorizations (just as it does with non-device objects).

- You should use the unique DCR device IDs supplied by the Device Credentials Repository Server available starting with CWCS 3.0.

For more information about DCR, see [Chapter 14, “Using the Device Credentials Repository”](#). The simplest way to do this is to ensure that:

- A DCR Server is running.
- The application data source has been populated with DCR IDs.
- The application OGS ASA you use to evaluate rules relating to device classes uses the DCR ID in the `"value"` parameter of the Object ID of every object that represents a device.

For example: If you are using the class `CMF:DCR:Device` to represent devices, and one of these devices has the DCR ID `123456`, your ASA must return the Object ID for this device as `CMF:DCR:Device$123456`.

If you do not do this, your Secure Views implementation will be unable to match any of entries in the CAM-returned list of authorized objects (which is a list of DCR IDs) against the `OGSObjectList`, and will not return any of them to the client (even though the user may be authorized to see them).

Note that you can work around the requirement of an ASA that preserves DCR IDs, as long as you can supply these IDs in another way (see the [“Using Secure Views Without DCR IDs” section on page 30-48](#)).

- You must ensure that the running instance of `OGSServerProxy` performing the filtering is running in the same Java VM as the CAM instance with which it communicates.

This restriction is imposed by the current implementation of CAM, which by default expects all of its clients to be servlets executing in the same servlet engine. CAM can be packaged with the process using `OGSServerProxy`.

## CISCO CONFIDENTIAL

- The default implementation of the filtering interface allows unrestricted access to members that do not represent devices. If your application needs additional filtering on non-device group members, you can either extend the default implementation or create a new implementation of the interface to meet your requirements. Be sure to indicate that you are using a new implementation, as explained in the [“Specifying a Non-Default Implementation”](#) section on page 30-48.

## Implementing Secure Views

Secure Views is intended for use in applications where:

- ACS mode is enabled.
- You intend to filter access to devices only.
- Devices are identified using device IDs supplied by the Device Credentials Repository (DCR IDs).

As long as your implementation observes the requirements explained in the [“How Secure Views Work”](#) section on page 30-43, using Secure Views is relatively simple, as explained in the following topics:

- [Installing Secure Views](#)
- [Using OGS SecurityContext](#)
- [Using Secure Views With DCR IDs](#)
- [Using Secure Views with Object Selector](#)
- [Using Secure Views With the OGS Administrative GUI](#)

The requirement for ACS mode is an absolute requirement. If your application cannot observe the requirements for device-only filtering and use of DCR IDs for devices, see the [“Customizing Your Secure Views Implementation”](#) section on page 30-47

## Installing Secure Views

To implement Secure Views, OGS 1.1 must be packaged with your application. You will need to extract the file `ogs1.1.war` from the OGS SDK, and to include the following packages in your build:

- `com.cisco.nm.xmls.ogs.client`
- `com.cisco.nm.xmls.ogs.util`
- `com.cisco.nm.xmls.ogs.server`

## Using OGS SecurityContext

The OGS interface specifies filtering of a group's membership based on the user/application/task context in which a request is made. Additionally, class `OGSSecurityContext` defines a constructor that accepts the user ID, application ID, and task ID (and, optionally, the session ID) that CAM requires to identify the devices for which the user has access.

If your application requires Secure Views, be sure that you are passing the proper information to this constructor before you make the request for group evaluation. If you do not do this, CAM will not have a chance to return the list of authorized devices before group evaluation is conducted.

The normal process for creating and using instances of `OGSSecurityContext` is:

- 
- Step 1** Create an instance of the class `TaskID` (in package `com.cisco.nm.xmls.ogs.util`) and assign the proper values to its attributes `TaskID`, `ApplicationID`, and `SessionID` (the `SessionID` can be null).

**CISCO CONFIDENTIAL**

- Step 2** Create an instance of OGSSecurityContext (in package com.cisco.nm.xml.ogs.util).
- Step 3** Pass the instance of the TaskID class to the OGSSecurityContext constructor.
- Step 4** Issue your group- or rule-evaluation request.
- 

**Using Secure Views With DCR IDs**

If you are using standard Object IDs that include the standard DCR ID for all group members that are devices, the reference OGSFilterImpl class and its DoFiltering method perform all of the following tasks automatically:

```
DoFiltering()
{
 Check if CiscoWorks is in ACS Mode;
 If CiscoWorks is not in ACS Mode, return OGSObjectList without filtering;
 If CiscoWorks is in ACS Mode;
 Get Admin Groups from CAM.
 Get Admin Device Groups that User is authorized to use for the given TaskID and
 ApplicationID
 Get the DCR ID of the devices in these Admin Device Groups;
 Avoid Filtering all non-device objects, using the property "DeviceClasses" in
 OGSClient.properties
 Get the Device Objects in the OGSObjectList passed to the function as a parameter;
 Get the Value portion of the Device Object. This is assumed to be the DCR ID;
 Intersect the list and return an OGSObjectList containing all non-device objects
 and only filtered device objects
}
```

**Using Secure Views with Object Selector**

As explained in the [“Creating an OGS GUI” section on page 30-40](#), integrating OGS with an Object Selector GUI is fairly simple. If your OGS implementation also includes Secure Views, however, you have a few extra tasks to perform:

- In the Action Class, which handles the form containing the Object Selector, your application must pass in the appropriate Application ID, Task ID, and Object Selector Name.
- Your application Tree Generator must retrieve the Task ID attribute and use it to create the OGSSecurityContext instance for the request.

As a reference, the default Tree Generator uses method getOGSUserContext to perform all the steps mentioned in the [“Using OGS SecurityContext” section on page 30-45](#). This includes:

- Creating an instance of the class TaskID with all the proper attribute values
- Creating an instance of OGSSecurityContext
- Passing the TaskID instance to the OGSSecurityContext constructor

To adapt your application Object Selector for use with Secure Views:

---

- Step 1** Set the Application ID into the HTTP Session.
- Step 2** Get an instance of OGSObjectSelectorAdapter.
- Step 3** Use the setObjectTaskIdList method to set the Task ID to the Task ID on which you want to filter.
- 

For example: Let us assume that your:



**CISCO CONFIDENTIAL**

- Object Selector Name is `FH.OGScontent` (this is the same as the `osname` specified in the `objectselector` JSP tag)
- Application ID is `iptm`.
- Task ID is `write_priv`.

Using these names, you would modify your Object Selector Action Class as follows:

```
public ActionForward perform (ActionMapping mapping)
 ActionForm form
 HttpServletRequest request
 HttpServletResponse response)
 throws IOException ServletException
{
 String YOUR-OBJECT-SELECTOR-NAME="FH.OGScontent"
 String YOUR-APPLICATION-ID="iptm"
 String YOUR-TASK-ID="write_priv"
 HttpSession session=request.getSession();
 session.setAttribute("AppID"+YOUR-OBJECT-SELECTOR-NAME, YOUR-APPLICATION-ID);
 OGSObjectSelectorAdapter osAdapter=new
 OGSObjectSelectorAdapter(request, "FH.OGScontent");
 osAdapter.setObjectTaskIdList("write_priv");
 ...
}
```

## Using Secure Views With the OGS Administrative GUI

As long as you have observed the limits described in the [“How Secure Views Work”](#) section on [page 30-43](#), your Secure Views implementation class will filter all device requests from all clients by default.

This can be a problem if your application uses the OGS Administrative GUI to allow users with sufficient authority to define groups. The OGS Client performing the filtering must be able to distinguish between normal client requests and those from an Administrative GUI, so that the Administrative GUI user has access to all devices when defining groups and their rules.

To distinguish them, make sure that the OGS Administrative GUI you have implemented in your application identifies itself to CAM with an Application ID of “OGS” and a TaskID of “OGSOPERATION” when issuing requests (these two IDs are case-sensitive and must be entered exactly as shown here).

These two values, when passed with `OGSSecurityContext`, turn off filtering functions in Secure Views implementations automatically.

If your application is used in a CiscoWorks suite, you may want to consider whether you want to implement the OGS Administrative GUI in your application at all.

CWCS 3.0 supplies the OGS Administrative GUI at the CiscoWorks level, with filtering turned off by default (that is, the CiscoWorks implementation of the OGS Administrative GUI already identifies itself with the Application ID “OGS” and the Task ID “OGSOPERATION”).

If you must implement a separate OGS Administrative GUI for your application, you will probably want to keep filtering turned on.

## Customizing Your Secure Views Implementation

It is possible to adapt Secure Views to some special requirements, including:

- [Specifying a Non-Default Implementation](#)
- [Using Secure Views Without DCR IDs](#)

## CISCO CONFIDENTIAL

### Specifying a Non-Default Implementation

In the default OGS packaging, the `OGSClientProperties` parameter `SecurityFilterImpl=` is explicitly set to `OGSFilterImpl`. This is the default or reference implementation of `OGSFilterIf` in `com.cisco.nm.xml.ogs.client`.

If you must handle special filtering requirements, you must either override the `OGSFilterImpl` implementation's `DoFilter ()` function or create a new class which implements `OGSFilterIf` (you are likely to do the latter if you are using Secure Views without DCR IDs, as explained in the [“Using Secure Views Without DCR IDs”](#) section on page 30-48.)

If you have created a new implementation of `OGSFilterIf`, you must set `SecurityFilterImpl` to the name of your new class. For example: `SecurityFilterImpl=MyOGSFilterImpl`.

### Using Secure Views Without DCR IDs

If your application does not use the DCR ID as the value of the Object ID for devices, you will need to create a new implementation of `OGSFilterIf` or extension of `OGSFilterImpl` that maps between whatever you are using in the Value portion of a Device Object and the actual DCR IDs.

#### Example

If you are using a Device Name for the Value portions of the Object ID for devices, you will need to create a new class with a `DoFiltering()` method that implements `OGSFilterIf`. You can do this mapping in any way that is convenient to you, but it must be performed before `OGSServerProxy` attempts to compare the CAM and OGS object lists:

```
DoFiltering()
{
 Check if CiscoWorks is in ACS Mode;
 If CiscoWorks is not in ACS Mode, return OGSObjectList without filtering;
 If CiscoWorks is in ACS Mode;
 Get Admin Device Groups from CAM.
 Get Admin Device Groups that User is authorized to use for the given TaskID and
 ApplicationID
 Get the DCR ID of the devices in these Admin Device Groups;
 Avoid filtering all non-device objects in the OGSObjectList returned by OGS Server
 Get the Device Objects in the OGSObjectList passed to the function as a parameter;
 Get the Value portion of the Device Object. This will be something other than DCRID;
 Map the OGS Value to DCRID and return the ObjectList with DCR IDs in Value position
 Intersect the list and return an OGSObjectList containing all non-device objects
 and only filtered device objects
```

**CISCO CONFIDENTIAL**

# Using OGS Common and Shared Groups

OGS Common Groups are groups created based on DCR attributes. They are propagated to other OGS Servers running on the same server and to peer servers in the same DCR cluster.

For example, the CWCS Common Groups will be propagated to the RME OGS Server, if RME is installed, so users of RME OGS Server will also be able to view these Common Groups.

OGS Shared Groups are application groups propagated to CWCS and other applications on the same server and to peer servers in the same DCR cluster.

For example, all Campus Manager groups are propagated to the RME OGS Server on the same server.

Common Groups have enumerated rules. Their membership is cached in a way similar to caching of a local group. Shared Groups are remote links to the respective OGSServer and are evaluated at runtime. Membership is not cached locally for Shared Groups, so group-membership change events are not sent for them.

A Provider Group refers to the root group name under each hierarchy.

For example, the Provider Group for a Common Services group hierarchy is `CS@hostname`, and for the RME hierarchy, it is `RME@hostname`. The hostname is appended to make the Provider Group unique, as Shared Groups are propagated across servers.

By default, the Provider Group name is of the format `application@hostname`. If the hostname changes, or the admin user configures the CWHP server name as the Provider Group suffix, the Provider Group name will be changed after a daemon restart.

For examples and User Interface of Common Groups and Shared Groups, see Chapter 5, Administering Groups, in User Guide for CiscoWorks Common Services.

Using Common and Shared Groups requires you to perform the tasks covered in the following topics:

- [Configuring OGSServer.properties](#)
- [Configuring SharedGroups.properties](#)
- [Implementing the SharedGroupObjectMapperIf Interface](#)

## Configuring OGSServer.properties

To enable Common and Shared Groups, applications should configure the following in `OGSServer.properties`:

```
Application OGS instances that do not wish to participate
in group sharing should remove the following property.
SharedGroupImplClass = com.cisco.nm.xml.ogs.sharedgroups.SharedGroupManager
```

## Configuring SharedGroups.properties

The following shows a generic `SharedGroups.properties` file, with explanation of the fields and how to set them.

```
Uncomment this line and specify a different value if your OGS
publishes its own URN
#LocalOgsURN = ogs_server_urn^M
#
Application OGS instances should uncomment the following line
#LocalSgURN = remote_ogs_urn
#
```

**CISCO CONFIDENTIAL**

```

Uncomment the following line and modify to match the URL of the CTMServlet
deployed by your application
#LocalURL = :443/<application webapp>/CTMServlet
#
SrcDeviceClass = :CMF:DCR:Device
#
Application OGS instances should specify their device class here
#
DestDeviceClass = <Application Device class>
#
The following should match the value of the property ProductId
of CMF-OGS
#
RemoteOgsProductId = CS
#
Uncomment the following property and specify the desired
depth of the MDF hierarchy if you wish to limit the depth
of the MDF hierarchy in common groups
#
#MDFHierarchyDepth = -1
ClientRegistryFile = clients.reg
#Application can uncomment the following to override the default rule for provider group
from CS hierarchy
#UseContainerForCommonGroups=true

```

## Implementing the SharedGroupObjectMapperIf Interface

To map application-specific object IDs to a common representation, applications should implement `SharedGroupObjectMapperIf`.

If your application uses the DCR device ID as the application object ID for OGS objects, you need only configure the `SrcDeviceClass` and `DestDeviceClass` in the `SharedGroups.properties` file. The default implementation (`com.cisco.nm.xml.ogs.sharedgroups.DefaultObjectMapper`) will handle the object mapping.

If you do not use DCR device IDs as application object IDs, you must implement `SharedGroupObjectMapperIf` and provide the implementation in the `SharedGroups.properties`, as follows:

```

#Object ID mapper class
ObjectIdMapClass = Customized object mapper class

```

The customized object mapper class should be in the classpath for the `OGSServer`.

## OGS Utility Class for Common and Shared Groups

The class `com.cisco.nm.ogs.util.OGSUtil` provides two APIs to get the common and local provider group name.

```

public static String getCommonRoot(); //for getting the CS root group
public static String getLocalRoot(); //for getting the local root group

```

**CISCO CONFIDENTIAL**

## Using OGS 1.3 Client Side Enhancements

Applications using OGS 1.3 have enhanced control over the display of GUI components. These client-side enhancements, which are not available with OGS 1.2, enable applications to determine:

- The display of some GUI components based on group types.
- The flow sequence of UI wizards based on group types.

Changes needed to support these client-side enhancements are available in the `com.cisco.nm.xms.ogs.client` package of the OGS module.

The following topics explain how to use these enhancements:

- [About the Enhanced OGS 1.3 Classes and Data Structures](#)
- [Controlling the Display of Wizard Steps](#)
- [Integrating OGS 1.3 With Your Application](#)

### About the Enhanced OGS 1.3 Classes and Data Structures

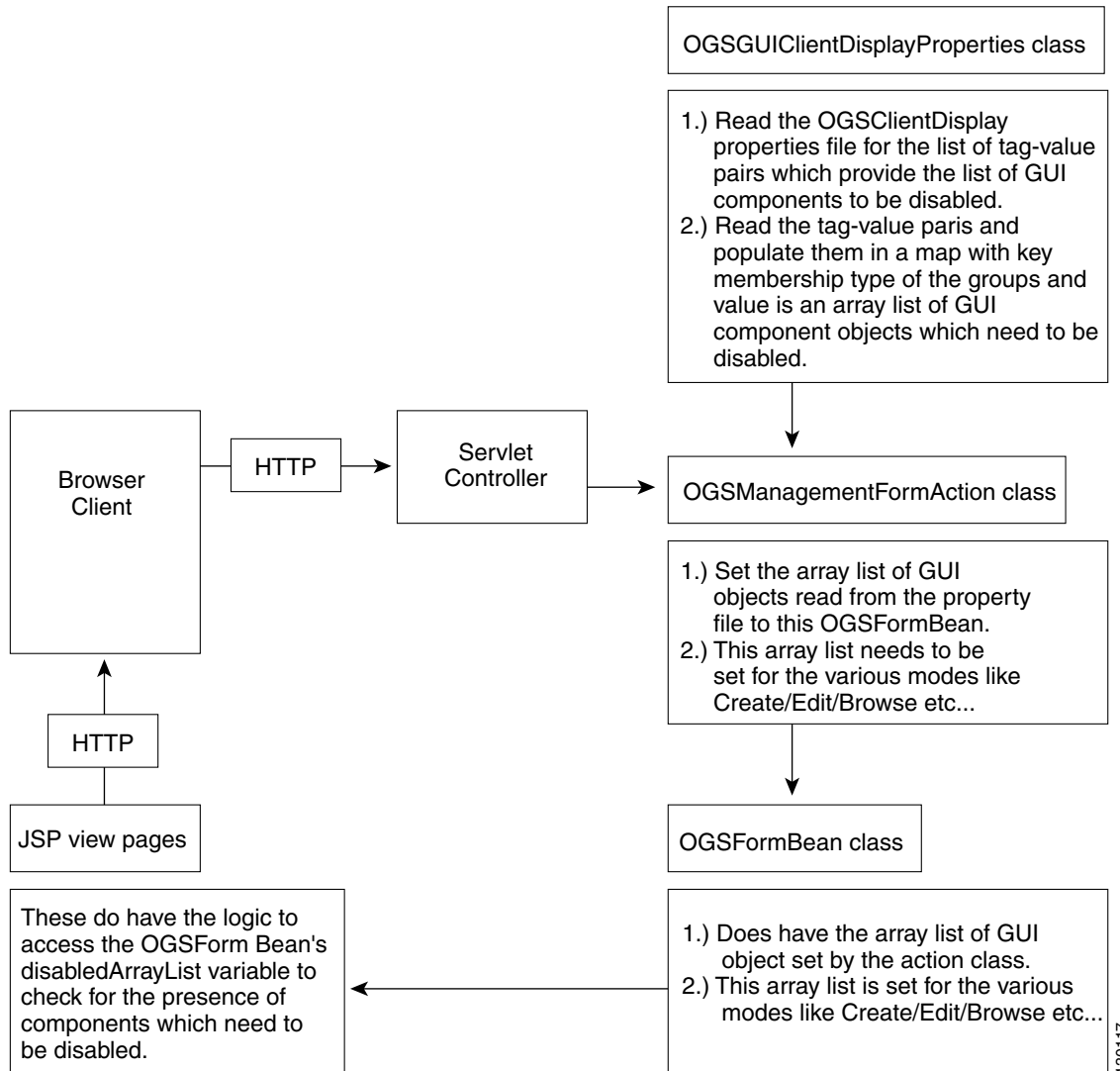
[Table 30-6](#) shows the OGS 1.3 classes enhanced to allow for GUI controls. [Figure 30-4](#) shows how these classes interact with each other and with the GUI components.

**Table 30-6** *Classes for Client Side Enhancements*

| Class                        | Description                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OGSClientUIDisplayProperties | Reads the tag-value pairs from <code>OGSClientUIDisplay.properties</code> , and stores the values in the hash map.                                                                                                                                                                                                                                                                                                      |
| OGSFormBean                  | Updated to store the Array List of GUI components which need to be disabled. Corresponding getter and setter methods are provided.                                                                                                                                                                                                                                                                                      |
| OGSManagementFormAction      | Updated with logic permitting the setting of the Array List of GUI components which have to be disabled. Based on the membership type of the group entries, the Form Bean is updated with the Array List of GUI objects.<br><br>Control of the display of the GUI components can be made specific to operational modes like Create, Edit and Browse.<br><br>You can use the new screen IDs to skip the membership page. |

**CISCO CONFIDENTIAL**

**Figure 30-4 OGS 1.3 Class and GUI Component Interactions**



The following JSP files were also modified to check for the availability of the GUI components in the disabled Array List:

- ogsBrowseProperties.jsp
- ogsCreateProperties.jsp
- ogsCreateRules.jsp
- ogsEditProperties.jsp
- ogsEditRules.jsp

Based on the presence of the components in the disabled Array List, the display of the view components can be controlled.

**CISCO CONFIDENTIAL**

## Controlling the Display of Wizard Steps

You need to create new screen IDs for controlling the display of wizard steps while in Create or Edit mode. The new screen IDs need to be defined in the site-map XML files of the respective web application. The screen IDs are:

- ogsSkipMemberCreateProperties
- ogsSkipMemberCreateRules
- ogsSkipMemberCreateSummary
- ogsSkipMemberEditProperties
- ogsSkipMemberEditRules
- ogsSkipMemberEditSummary

Configure the screen IDs to skip the Membership page during the Create and Edit wizard tasks of the group entries. However, support is not provided for skipping Rules, Properties and Summary pages, as these are required for group handling.

**Note**

Source code is available in /vob/enm\_ogs/share/classes/client/com/cisco/nm/ms/ogs/client, under the ogs1\_3 view.

## Integrating OGS 1.3 With Your Application

To integrate OGS 1.3's enhanced client-side features with your application:

- 
- Step 1** Include all OGS 1.3-related jar files with the application.
- Although all the changes were made in the ogs-client1.3.jar, we suggest that you refresh all jar files, to ensure that you have uniform versions throughout.
- Step 2** Update the OGSClientUIDisplay.properties file with the requisite entries.
- For a complete set of entries related to UI components, which you can configure, see [Example 30-2 on page 30-55](#). This file can be placed under the WEB-INF/classes directory of the respective web application, under Tomcat.
- Step 3** Define the TAGSTABLE definition in the XML file representing the Group Entries. Similarly, define the PROPERTYTABLE with the PROPERTY KEY as MEMBERSHIP\_TYPE and the desired value.
- For an example system-groups.xml file, see [Example 30-3 on page 30-56](#). The file is currently available under the WEB-INF/classes directory of the respective web-app under Tomcat servlet.
- Step 4** Make sure the string value provided for the MEMBERSHIP\_TYPE key matches the value provided in the tag of the OGSClientUIDisplay.properties.

If the XML file has entries like these for representing Cisco Access Servers:

```
<GROUP>
 <NAME>/System Defined Groups/Cisco Access Servers</NAME>
 <DESCRIPTION>Group for Cisco Access Servers</DESCRIPTION>
 <TYPE value="DYNAMIC" />
 <OGS_RULE_CONTAINER>
 <OGSRULE> </OGSRULE>
 </OGS_RULE_CONTAINER>
 <TAGSTABLE ref="devicegroupuptags" />
 <READ_PERMISSION_LIST ref="read_user_list" />
```

**CISCO CONFIDENTIAL**

```

 <WRITE_PERMISSION_LIST ref="write_user_list" />
 <EVALUATE_PERMISSION_LIST ref="evaluate_user_list" />
</GROUP>
<PROPERTYTABLE name="devicegroupptags">
 <PROPERTY>
 <KEY>MEMBERSHIP_TYPE</KEY>
 <VALUE>DEVICE</VALUE>
 </PROPERTY>
</PROPERTYTABLE>

```

Then the OGSCClientUIDisplay.properties file can have entries like:

```
DEVICE.privateVisibility=disabled
```

The string “DEVICE” in the properties file needs to match the string defined in the XML file. However applications can use any convenient string that is relevant in their application space.

- Step 5** Define the entries related to the new screen in the struts-config.xml and site-map XMLfiles for the respective webapp.

The entries in struts-config.xml are:

```

forward name="ogsSkipMemberCreateProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberCreateProperties" />
 <forward name="ogsSkipMemberCreateRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberCreateRules" />
 <forward name="ogsSkipMemberCreateSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberCreateSummary" />
 <forward name="ogsSkipMemberEditProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberEditProperties" />
 <forward name="ogsSkipMemberEditRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberEditRules" />
 <forward name="ogsSkipMemberEditSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberEditSummary" />

```

The entries in site-map.xml are:

```

<appWizard>
 <appWizardItem>
 <label>Properties</label>
 <appContentArea screenID="ogsSkipMemberCreateProperties"
contentAreaTitle="Properties"
helpTag="group_administration"
fileRef="/WEB-INF/screens/OGS/ogsCreateProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsSkipMemberCreateRules"
contentAreaTitle="Rules"
helpTag="group_administration"
fileRef="/WEB-INF/screens/OGS/ogsCreateRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsSkipMemberCreateSummary"
contentAreaTitle="Summary"
helpTag="group_administration"
fileRef="/WEB-INF/screens/OGS/ogsCreateSummary.jsp" />
 </appWizardItem>
</appWizard>

<appWizard>
 <appWizardItem>
 <label>Properties</label>

```



**CISCO CONFIDENTIAL**

```

 <appContentArea screenID="ogsSkipMemberEditProperties"
 contentAreaTitle="Properties"
 helpTag="group_administration"
 fileRef="/WEB-INF/screens/OGS/ogsEditProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsSkipMemberEditRules"
 contentAreaTitle="Rules"
 helpTag="group_administration"
 fileRef="/WEB-INF/screens/OGS/ogsEditRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsSkipMemberEditSummary"
 contentAreaTitle="Summary"
 helpTag="group_administration"
 fileRef="/WEB-INF/screens/OGS/ogsEditSummary.jsp" />
 </appWizardItem>
</appWizard>

```

**Example 30-2 OGSClientUIDisplay.properties**

```

Properties file for the OGS Client UI display
This file is mainly used to control the display of UI components based on group types
and is valid only for OGS Admin GUI.
It is assumed that all the OGS GUI components are enabled by default.
The respective components along with the group types for which they need to be disabled
can be mentioned here.

Copyright (c) 2004 Cisco Systems, Inc.
All rights reserved
#
THIS SOFTWARE IS THE PROPERTY OF CISCO SYSTEMS INC.
2004
THE RECIPIENT BY ACCEPTING THIS MATERIAL AGREES THAT THE
MATERIAL WILL NOT BE USED, COPIED OR REPRODUCED IN WHOLE OR
IN PART NOR ITS CONTENTS REVEALED IN ANY MANNER WITHOUT THE
EXPRESS WRITTEN PERMISSION OF CISCO SYSTEMS, INC.

#
The names which refer to the various components are mentioned below.
creategroupSelect -> Group Select button in Properties:Create page
createparentChange -> Change Parent button in Properties:Create page
DynamicEvaluation -> Radio button for Automatic
StaticEvaluation -> Radio button for Based Upon User Request
publicVisibility ->Visibility scope radio button "Available to all users"
privateVisibility -> Visibility scope radio button "Available to created user only"
createORoperator -> OR operator in the Rules:Create page
createANDoperator -> AND operator in the Rules:Create page
createEXCLUDEoperator -> EXCLUDE operator in the Rules:Create page
createMembershipWizard ->Disabling the Membership page step in the Creation Wizard
editORoperator -> OR operator in the Rules:Edit page
editANDoperator -> AND operator in the Rules:Edit page
editEXCLUDEoperator -> EXCLUDE operator in the Rules:Edit page
editMembershipWizard ->Refers to the Membership page step in the Edition Wizard
browseViewParentRule -> View Parent Rule button related to Properties:Details page
browseMembershipDetails -> Membership Details button related to Properties:Details page

```

**CISCO CONFIDENTIAL**

```

EditGroupName -> Text field related to the Group Name in the Edit:Properties page
The KEY value contains a string separated by "."
The first token of the key should match the value given for the MEMBERSHIP_TYPE in the
PROPERTY_TABLE pertaining to XML file containing the group definition.
Sample examples are given below:
#
#DEVICE.createGroupSelect=disabled
#DEVICE.createParentChange=disabled
DEVICE.DynamicEvaluation=disabled
#DEVICE.StaticEvaluation=disabled
DEVICE.privateVisibility=disabled
DEVICE.publicVisibility=disabled
DEVICE.createAddRuleExpression=disabled
DEVICE.createSyntaxCheck=disabled
DEVICE.createViewParentRule=disabled
DEVICE.createORoperator=disabled
DEVICE.createANDoperator=disabled
DEVICE.createEXCLUDEoperator=disabled
DEVICE.editORoperator=disabled
DEVICE.editANDoperator=disabled
DEVICE.editEXCLUDEoperator=disabled
DEVICE.editAddRuleExpression=disabled
DEVICE.editSyntaxCheck=disabled
DEVICE.editViewParentRule=disabled
DEVICE.createMembershipWizard=disabled
DEVICE.editMembershipWizard=disabled
INTERFACE.createGroupSelect=disabled
INTERFACE.createParentChange=disabled

```

---

**Example 30-3 System-Groups.xml**

```

<?xml version="1.0"?>
<!DOCTYPE SYSTEMGROUPS [
<!ELEMENT SYSTEMGROUPS (GROUP+,PROPERTYTABLE+,USERLIST+) >
<!ELEMENT GROUP
(NAME,DESCRIPTION?,TYPE,OGS_RULE_CONTAINER,TAGSTABLE?,READ_PERMISSION_LIST,WRITE_PERMISSIO
N_LIST,EVALUATE_PERMISSION_LIST)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT TYPE EMPTY>
<!ATTLIST TYPE value (DYNAMIC|STATIC) #REQUIRED >
<!ELEMENT OGS_RULE_CONTAINER (OGSRULE?)>
<!ELEMENT OGSRULE (#PCDATA) >
<!ELEMENT TAGSTABLE EMPTY>
<!ATTLIST TAGSTABLE ref CDATA #REQUIRED >
<!ELEMENT READ_PERMISSION_LIST EMPTY>
<!ATTLIST READ_PERMISSION_LIST ref CDATA #REQUIRED >
<!ELEMENT WRITE_PERMISSION_LIST EMPTY>
<!ATTLIST WRITE_PERMISSION_LIST ref CDATA #REQUIRED >
<!ELEMENT EVALUATE_PERMISSION_LIST EMPTY>
<!ATTLIST EVALUATE_PERMISSION_LIST ref CDATA #REQUIRED >
<!ELEMENT PROPERTYTABLE (PROPERTY+)>
<!ATTLIST PROPERTYTABLE
name CDATA #REQUIRED >
<!ELEMENT PROPERTY (KEY,VALUE)>
<!ELEMENT KEY (#PCDATA)>
<!ELEMENT VALUE (#PCDATA)>
<!ELEMENT USERLIST (USER*)>
<!ATTLIST USERLIST
name CDATA #REQUIRED >

```

**CISCO CONFIDENTIAL**

```

<!ELEMENT USER (#PCDATA)>
]>
<SYSTEMGROUPS>
<GROUP>
 <NAME>/System Defined Groups/Unknown Devices</NAME>
 <DESCRIPTION>Group for devices whose MDFid is unknown</DESCRIPTION>
 <TYPE value="DYNAMIC" />
 <OGS_RULE_CONTAINER>
 <OGSRULE>:CMF:DCR:Device.MDFid equals "UNKNOWN" </OGSRULE>
 </OGS_RULE_CONTAINER>
 <TAGSTABLE ref="devicegroupstags" />
 <READ_PERMISSION_LIST ref="read_user_list" />
 <WRITE_PERMISSION_LIST ref="write_user_list" />
 <EVALUATE_PERMISSION_LIST ref="evaluate_user_list" />
</GROUP>
<GROUP>
 <NAME>/System Defined Groups/Cisco Access Servers</NAME>
 <DESCRIPTION>Group for Cisco Access Servers</DESCRIPTION>
 <TYPE value="DYNAMIC" />
 <OGS_RULE_CONTAINER>
 <OGSRULE> </OGSRULE>
 </OGS_RULE_CONTAINER>
 <TAGSTABLE ref="devicegroupstags" />
 <READ_PERMISSION_LIST ref="read_user_list" />
 <WRITE_PERMISSION_LIST ref="write_user_list" />
 <EVALUATE_PERMISSION_LIST ref="evaluate_user_list" />
</GROUP>
<GROUP>
 <NAME>/System Defined Groups/Cisco Cable Devices</NAME>
 <DESCRIPTION>Group for Cisco Cable Devices</DESCRIPTION>
 <TYPE value="DYNAMIC" />
 <OGS_RULE_CONTAINER>
 <OGSRULE> </OGSRULE>
 </OGS_RULE_CONTAINER>
 <TAGSTABLE ref="interfacegroupstags" />
 <READ_PERMISSION_LIST ref="read_user_list" />
 <WRITE_PERMISSION_LIST ref="write_user_list" />
 <EVALUATE_PERMISSION_LIST ref="evaluate_user_list" />
</GROUP>
<GROUP>
 <NAME>/System Defined Groups/Cisco Cable Devices/Cisco 6900 Series Multiplexers</NAME>
 <DESCRIPTION>Group for Cisco 6900 Series Multiplexers</DESCRIPTION>
 <TYPE value="DYNAMIC" />
 <OGS_RULE_CONTAINER>
 <OGSRULE> </OGSRULE>
 </OGS_RULE_CONTAINER>
 <TAGSTABLE ref="interfacegroupstags" />
 <READ_PERMISSION_LIST ref="read_user_list" />
 <WRITE_PERMISSION_LIST ref="write_user_list" />
 <EVALUATE_PERMISSION_LIST ref="evaluate_user_list" />
</GROUP>
<PROPERTYTABLE name="systemdefinedgroupstags">
 <PROPERTY>
 <KEY>SYSTEM_PREDEFINED</KEY>
 <VALUE>TRUE</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="customizablegroupstags">
 <PROPERTY>
 <KEY>CUSTOMIZABLE_GROUP</KEY>
 <VALUE>TRUE</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="hiddengroupstags">

```

**CISCO CONFIDENTIAL**

```

<PROPERTY>
 <KEY>HIDDEN</KEY>
 <VALUE>TRUE</VALUE>
</PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="devicegroupstags">
 <PROPERTY>
 <KEY>MEMBERSHIP_TYPE</KEY>
 <VALUE>DEVICE</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="interfacegroupstags">
 <PROPERTY>
 <KEY>MEMBERSHIP_TYPE</KEY>
 <VALUE>INTERFACE</VALUE>
 </PROPERTY>
</PROPERTYTABLE>

```

---

## Using OGS 1.4 Enhancements

The following enhancements are part of OGS 1.4 release:

### Template Based Groups

Template based groups simplify group creation. You can create frequently used groups easily. If you want to create a group based on a device attribute (such as Location) you must select the Location based groups template, and enter the location values for the rule.

However, for group creation with more than one attribute (such as Location and IP Address) you must use the regular Rule based group creation.

### Group ID

Group ID is introduced to the OGSGroupDefinition (com.cisco.nm.xml.ogs.util) class. Group ID is unique for a particular group. However, when a group is deleted, and then recreated, the group will not have the same group ID. A Getter method (groupId()) is added to the OGSGroupDefinition class for the applications to use the group ID.

Only the evaluateGroup method in the OGSInterface is overloaded to use the group ID, instead of the group name. The others such as deleteGroup and copyGroup continue to use group name.

The evaluateGroup() API is used from the application backend, and require the group ID.

A new class – com.cisco.nm.xml.ogs.util.OGSGroupId is added. This will encapsulate the group ID of a particular group as a string.

### Configurable Display Name for Class Names

In group administration UIs, the internal class names (:CMF:DCR:Device or Kilner:VASA:KilnerObject:Device:Host) are displayed. This is not very useful for the end users.

With configurable display name for class names, you can have more user-friendly UI display names (such as Device or Host).

OGSClient.properties contains the mapping definitions. Applications can enable this feature using a specified property (EnableClassNameMapping=true in OGSClient.properties), and provide the mapping from the internal class name to the external class name.

## CISCO CONFIDENTIAL

### Configurable Root (Provider) Group

In a single-server setup (DCR in standalone mode), the root group name is of the format *Application*. In a multi-server setup (DCR in Master / Slave mode), the root group name is of the format *Application-CWHP Display Name*.

Both the formats are available as templates in a configuration file, *NMSROOT/objects/groups/RootGroupTemplate.properties*. The default content of the file is:

```
This property file is for configuring the template to be used for
root (provider) group of all group hierarchies in this server
ProviderTemplateForSingleServer = $ProductId$
ProviderTemplateForMultiServer = "$ProductId$-$CWHPServerName$"
```

When the CiscoWorks administrator changes the DCR mode from standalone to master/slave, the root (provider) group name of all group hierarchies will be changed according to the template.

This change will be done automatically, and does not need the daemons to be restarted. Similarly, the root group name will be changed when the DCR mode is changed from Master/Slave to Standalone.

In a multi-server setup, the template can be changed to *\$CWHPServerName\$ - \$ProductId\$*. This change is global, applies to all OGS servers in the CiscoWorks machine, and takes effect only after the daemons are restarted. Hence, the properties file is shipped by Common Services and there will be only one properties file in a CiscoWorks installation.

The Common Services root group hierarchy will always be CS, irrespective of whether the server is in single-server mode or multi-server mode. Since there will be only one CS group hierarchy in a multi-server setup, there is no need to uniquely identify the CS group hierarchy by adding the CWHP display name.

`com.cisco.nm.xml.ogs.util.OGSUtil` will be augmented with the following APIs:

- Public string `getCommonRoot()`—Returns the common group root (root of the CS group hierarchy). The value, as explained, will be CS.
- Public string `getLocalRoot()`—Returns the current root group name of the local hierarchy. For example, this API will return the RME root group name when called with the classpath of the `RMEOGSServer`.
- Public string `getOldLocalRoot()`—Returns the old root group name of the local hierarchy.

## Integrating OGS 1.4 With Your Application

To integrate the client-side features of OGS 1.4 with your application:

- 
- Step 1** Include all OGS 1.4 related jar files in the applications and ensure that they are shipped with the respective webapp.
- Step 2** Update the `OGSClient.properties` file with the following entries in the respective webapp environment:

```
TemplateSupportEnabled = true
TemplateFile=\\cwhp\\WEB-INF\\classes\\Templates.ser
DefaultTemplateToBeSelected=<Template String which needs to be selected default>
```

(\\Indicates the relative location of the path from webapps directory).

The default location of `OGSClient.properties` file is the `WEB-INF/classes` directory, relative to `MDC\tomcat\webapps\app-name`.

By default, the template support is disabled because it is not consumed by all applications.

- Step 3** Configure an XML file that describes the templates.

**CISCO CONFIDENTIAL**

The DTD for the XML file is provided here, along with sample entries. Applications should specify the entries accordingly so that they would be made available out of the box.

**Note**

Skipping the Membership page and enabling template support cannot be done together for the same group type. However, backward compatibility will be maintained for the other group types for which template support is not enabled.

Sample entries for creating templates in a XML file. Here the ITM application has been taken as an example for integration.

**DTD for the XML File**

```
<?xml version="1.0"?>

<!--*****-->
<!-- Copyright (c) 2005 Cisco Systems, Inc. -->
<!-- All rights reserved. -->
<!--*****-->

<!DOCTYPE TEMPLATES [
<!ELEMENT TEMPLATES (TEMPLATE+,PROPERTYTABLE+) >
<!ELEMENT TEMPLATE (NAME,DESCRIPTION?,TEMPLATE_RULE_CONTAINER,DISPLAY_LABEL?,TAGSTABLE?)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT TEMPLATE_RULE_CONTAINER (TEMPLATERULE?)>
<!ELEMENT TEMPLATERULE (#PCDATA) >
<!ELEMENT DISPLAY_LABEL (#PCDATA)>
<!ELEMENT TAGSTABLE EMPTY>
<!ATTLIST TAGSTABLE ref CDATA #REQUIRED >
<!ELEMENT PROPERTYTABLE (PROPERTY+)>
<!ATTLIST PROPERTYTABLE name CDATA #REQUIRED >
<!ELEMENT PROPERTY (KEY,VALUE)>
<!ELEMENT KEY (#PCDATA)>
<!ELEMENT VALUE (#PCDATA)>
]>
<TEMPLATES>
<TEMPLATE>
 <NAME>Location Based template</NAME>
 <DESCRIPTION>This template is used for creating groups based upon
location</DESCRIPTION>
 <TEMPLATE_RULE_CONTAINER>
 <TEMPLATERULE>:ITM:VASA:ITMObject:Device.Location contains</TEMPLATERULE>
 </TEMPLATE_RULE_CONTAINER>
 <DISPLAY_LABEL>List of Locations</DISPLAY_LABEL>
 <TAGSTABLE ref="Locationtemplates"/>
</TEMPLATE>
<TEMPLATE>
 <NAME>Name Based template</NAME>
 <DESCRIPTION>This template is used for creating groups based upon Name</DESCRIPTION>
 <TEMPLATE_RULE_CONTAINER>
 <TEMPLATERULE>:ITM:VASA:ITMObject:Device.Name contains</TEMPLATERULE>
 </TEMPLATE_RULE_CONTAINER>
 <DISPLAY_LABEL>List of MDFids</DISPLAY_LABEL>
 <TAGSTABLE ref="Nametemplates"/>
</TEMPLATE>
<TEMPLATE>
 <NAME>Subnet Based template</NAME>
 <DESCRIPTION>This template is used for creating groups based upon Subnet</DESCRIPTION>
 <TEMPLATE_RULE_CONTAINER>
 <TEMPLATERULE>:ITM:VASA:ITMObject:Device.IP.Netmask contains</TEMPLATERULE>
 </TEMPLATE_RULE_CONTAINER>
```

**CISCO CONFIDENTIAL**

```

 <DISPLAY_LABEL>List of Subnets</DISPLAY_LABEL>
 <TAGSTABLE ref="Subnettemplates"/>
</TEMPLATE>
<TEMPLATE>
 <NAME>Service Based template</NAME>
 <DESCRIPTION>This template is used for creating groups based upon
Service</DESCRIPTION>
 <TEMPLATE_RULE_CONTAINER>
 <TEMPLATERULE>:ITM:VASA:ITMObject:Device.Type contains</TEMPLATERULE>
 </TEMPLATE_RULE_CONTAINER>
 <DISPLAY_LABEL>List of Series</DISPLAY_LABEL>
 <TAGSTABLE ref="Servicetemplates"/>
</TEMPLATE>
<TEMPLATE>
 <NAME>Model Based template</NAME>
 <DESCRIPTION>This template is used for creating groups based upon Model</DESCRIPTION>
 <TEMPLATE_RULE_CONTAINER>
 <TEMPLATERULE>:ITM:VASA:ITMObject:Device.Model contains</TEMPLATERULE>
 </TEMPLATE_RULE_CONTAINER>
 <DISPLAY_LABEL>List of Models</DISPLAY_LABEL>
 <TAGSTABLE ref="Modeltemplates"/>
</TEMPLATE>

<PROPERTYTABLE name="Locationtemplates">
 <PROPERTY>
 <KEY>ATTRIBUTE</KEY>
 <VALUE>Location</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="Nametemplates">
 <PROPERTY>
 <KEY>ATTRIBUTE</KEY>
 <VALUE>Name</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="Servicetemplates">
 <PROPERTY>
 <KEY>ATTRIBUTE</KEY>
 <VALUE>Type</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="Modeltemplates">
 <PROPERTY>
 <KEY>ATTRIBUTE</KEY>
 <VALUE>Model</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name="Subnettemplates">
 <PROPERTY>
 <KEY>ATTRIBUTE</KEY>
 <VALUE>IP.NetMask</VALUE>
 </PROPERTY>
</PROPERTYTABLE>
<PROPERTYTABLE name=" ">
 <PROPERTY>
 <KEY>ATTRIBUTE</KEY>
 <VALUE></VALUE>
 </PROPERTY>
</PROPERTYTABLE>

</TEMPLATES>

```

**CISCO CONFIDENTIAL**

**Note** The element TAGSTABLE is optional. TAGSTABLE is to add tags if the application needs to find the groups created based on a specific template.

**Step 4** Define entries related to the new screen IDs in struts-config.xml file and site-map XML file of the respective webapp.

**Entries for struts-config.xml**

The following entries must be added in the OGS related sections of struts-config.xml.

```
<forward name="ogsBrowseProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsBrowseProperties" />
 <forward name="ogsBrowseRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsBrowseRules" />
 <forward name="ogsBrowseMembership"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsBrowseMembership" />
 <forward name="ogsBrowseSubgroups"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsBrowseSubgroups" />
 <forward name="ogsCreateProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsCreateProperties" />
 <forward name="ogsTemplateCreateProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateCreateProperties" />
 <forward name="ogsCreateRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsCreateRules" />
 <forward name="ogsDefaultCreateRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsDefaultCreateRules" />

<forward name="ogsTemplateCreateRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateCreateRules" />
 <forward name="ogsCreateMembership"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsCreateMembership" />
 <forward name="ogsDefaultCreateMembership"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsDefaultCreateMembership" />

<forward name="ogsTemplateCreateMembership"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateCreateMembership" />
 <forward name="ogsCreateAccessControl"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsCreateAccessControl" />
 <forward name="ogsCreateSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsCreateSummary" />
 <forward name="ogsDefaultCreateSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsDefaultCreateSummary" />

<forward name="ogsTemplateCreateSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateCreateSummary" />
 <forward name="ogsEditProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsEditProperties" />

<forward name="ogsTemplateEditProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateEditProperties" />
 <forward name="ogsEditRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsEditRules" />
 <forward name="ogsTemplateEditRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateEditRules" />
 <forward name="ogsEditMembership"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsEditMembership" />
 <forward name="ogsTemplateEditMembership"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateEditMembership" />
 <forward name="ogsEditAccessControl"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsEditAccessControl" />
```



**CISCO CONFIDENTIAL**

```

 <forward name="ogsEditSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsEditSummary" />
 <forward name="ogsTemplateEditSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsTemplateEditSummary" />
 <forward name="ogsMainSetup"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsMainSetup" />
 <forward name="ogsRoleAccessControl"
path="/WEB-INF/screens/popup.jsp?sid=ogsRoleAccessControl" />

<forward name="ogsServerError" path="/WEB-INF/screens/uii/index.jsp?sid=ogsServerError" />

<forward name="ogsSkipMemberCreateProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberCreateProperties" />

<forward name="ogsSkipMemberCreateRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberCreateRules" />

<forward name="ogsSkipMemberCreateSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberCreateSummary" />

<forward name="ogsSkipMemberEditProperties"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberEditProperties" />

<forward name="ogsSkipMemberEditRules"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberEditRules" />

<forward name="ogsSkipMemberEditSummary"
path="/WEB-INF/screens/uii/index.jsp?sid=ogsSkipMemberEditSummary" />

```

**Entries for Sitemap XML File**

The following entries must be added in the OGS-related section in the sitemap XML file of the respective webapp:

```

<?xml version="1.0"?>
<contentAreaSet>
 <navContentArea screenID="ogsMainSetup"
 contentAreaTitle="Group Administration"
 helpTag="group_administration"
 fileRef="/WEB-INF/screens/OGS/ogsMainSetup.jsp">
 </navContentArea>
 <appContentArea screenID="ogsBrowseProperties"
 contentAreaTitle="Property Details"
 helpTag="cs_groups_admin_view"
 fileRef="/WEB-INF/screens/OGS/ogsBrowseProperties.jsp" />
 <appContentArea screenID="ogsBrowseMembership"
 contentAreaTitle="Membership Details"
 helpTag="cs_groups_admin_view"
 fileRef="/WEB-INF/screens/OGS/ogsBrowseMembership.jsp" />
 <appContentArea screenID="ogsServerError"
 contentAreaTitle="Group Administration Server Error"
 helpTag="ogs_server_error"
 fileRef="/WEB-INF/screens/OGS/serverError.jsp" />
 <appContentArea screenID="ogsDCRSecondary"
 contentAreaTitle=""
 helpTag="ogs_dcr_secondary"
 fileRef="/WEB-INF/screens/OGS/ogsDCRSecondary.jsp" />
 <appContentArea screenID="ogsNewPage"
 contentAreaTitle=""
 helpTag="ogs_dcr_secondary"
 fileRef="/WEB-INF/screens/OGS/ogsNewPage.jsp" />

<appWizard>
 <appWizardItem>

```

**CISCO CONFIDENTIAL**

```

<label>Properties</label>
<appContentArea screenID="ogsTemplateCreateProperties"
 contentAreaTitle="Properties"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsTemplateCreateProperties.jsp" />
</appWizardItem >
<appWizard>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsDefaultCreateRules"
 contentAreaTitle="Rules"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsCreateRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Membership</label>
 <appContentArea screenID="ogsDefaultCreateMembership"
 contentAreaTitle="Membership"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsCreateMembership.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsDefaultCreateSummary"
 contentAreaTitle="Summary"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsCreateSummary.jsp" />
 </appWizardItem>
</appWizard>
 <appWizard>
 <appWizardItem>
 <label>Templates</label>
 <appContentArea screenID="ogsTemplateCreateRules"
 contentAreaTitle="Templates"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsTemplateCreateRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Membership</label>
 <appContentArea screenID="ogsTemplateCreateMembership"
 contentAreaTitle="Membership"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsCreateMembership.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsTemplateCreateSummary"
 contentAreaTitle="Summary"
 helpTag="cs_groups_admin_create"
 fileRef="/WEB-INF/screens/OGS/ogsCreateSummary.jsp" />
 </appWizardItem>
 </appWizard>
</appWizard>
<appWizard>
 <appWizardItem>
 <label>Properties</label>
 <appContentArea screenID="ogsEditProperties"
 contentAreaTitle="Properties"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsEditRules"

```

**CISCO CONFIDENTIAL**

```

 contentAreaTitle="Rules"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditRules.jsp" />
 </appWizardItem>
</appWizardItem>
 <label>Membership</label>
 <appContentArea screenID="ogsEditMembership"
 contentAreaTitle="Membership"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditMembership.jsp" />
</appWizardItem>
</appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsEditSummary"
 contentAreaTitle="Summary"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditSummary.jsp" />
</appWizardItem>
</appWizard>

<appWizard>
 <appWizardItem>
 <label>Properties</label>
 <appContentArea screenID="ogsCreateProperties"
 contentAreaTitle="Properties"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsCreateProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsCreateRules"
 contentAreaTitle="Rules"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsCreateRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Membership</label>
 <appContentArea screenID="ogsCreateMembership"
 contentAreaTitle="Membership"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsCreateMembership.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsCreateSummary"
 contentAreaTitle="Summary"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsCreateSummary.jsp" />
 </appWizardItem>
</appWizard>

<appWizard>
 <appWizardItem>
 <label>Properties</label>
 <appContentArea screenID="ogsTemplateEditProperties"
 contentAreaTitle="Properties"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Templates</label>
 <appContentArea screenID="ogsTemplateEditRules"
 contentAreaTitle="Templates"
 helpTag="cs_groups_admin_modify"

```

**CISCO CONFIDENTIAL**

```

 fileRef="/WEB-INF/screens/OGS/ogsTemplateEditRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Membership</label>
 <appContentArea screenID="ogsTemplateEditMembership"
 contentAreaTitle="Membership"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditMembership.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsTemplateEditSummary"
 contentAreaTitle="Summary"
 helpTag="cs_groups_admin_modify"
 fileRef="/WEB-INF/screens/OGS/ogsEditSummary.jsp" />
 </appWizardItem>
</appWizard>

<appWizard>
 <appWizardItem>
 <label>Properties</label>
 <appContentArea screenID="ogsSkipMemberCreateProperties"
 contentAreaTitle="Properties"
 helpTag=" "
 fileRef="/WEB-INF/screens/OGS/ogsCreateProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsSkipMemberCreateRules"
 contentAreaTitle="Rules"
 helpTag=" "
 fileRef="/WEB-INF/screens/OGS/ogsCreateRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsSkipMemberCreateSummary"
 contentAreaTitle="Summary"
 helpTag=" "
 fileRef="/WEB-INF/screens/OGS/ogsCreateSummary.jsp" />
 </appWizardItem>
</appWizard>

 <appWizard>
 <appWizardItem>
 <label>Properties</label>
 <appContentArea screenID="ogsSkipMemberEditProperties"
 contentAreaTitle="Properties"
 helpTag=" "
 fileRef="/WEB-INF/screens/OGS/ogsEditProperties.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Rules</label>
 <appContentArea screenID="ogsSkipMemberEditRules"
 contentAreaTitle="Rules"
 helpTag=" "
 fileRef="/WEB-INF/screens/OGS/ogsEditRules.jsp" />
 </appWizardItem>
 <appWizardItem>
 <label>Summary</label>
 <appContentArea screenID="ogsSkipMemberEditSummary"
 contentAreaTitle="Summary"
 helpTag=" "
 fileRef="/WEB-INF/screens/OGS/ogsEditSummary.jsp" />
 </appWizardItem>
</appWizard>

```

**CISCO CONFIDENTIAL**

```
</appWizard>
</contentAreaSet>
```

**Note**

You need to make these changes to the struts-config.xml and respective sitemap XML files only if template support is required. If template support is not required, you do not need to make any changes to the existing files.

**Entry for Displaying Error Messages**

Add the following entry in the appropriate property file for the webapp to display error messages:

```
ogs.noTemplateAttrValues=<Attribute value>
```

You must enter at least one value for this entry.

**Generating Serialized File from the XML File**

To generate a serialized file from the XML file, enter the following (where? CLI?):

```
com.cisco.nm.xms.ogs.client.templates.OGSTemplateGenerator XML_file_name
serialized_file_name
```

To run this command (?), the following files should be available in the classpath:

```
NMSROOT\MDC\tomcat\shared\lib\mice.jar;
```

```
NMSROOT\www\classpath;
```

```
NMSROOT\lib\classpath;
```

```
NMSROOT\MDC\tomcat\shared\lib\MICE.jar;
```

```
NMSROOT\lib\classpath\jconn2.jar;
```

```
NMSROOT\MDC\tomcat\webapps\cwhp\WEB-INF\lib\log4j.jar;
```

```
NMSROOT\MDC\tomcat\shared\lib\xerces.jar;
```

```
NMSROOT\MDC\tomcat\shared\lib\xalan.jar;
```

```
NMSROOT\MDC\tomcat\webapps\cwhp\WEB-INF\lib\ogs-client1.2.jar;
```

```
NMSROOT\MDC\tomcat\webapps\cwhp\WEB-INF\lib\cogs1.2.jar;
```

## Integrating Configurable Display Name for Class Names Feature with Applications

In OGSCient.properties, the following properties are added to enable this feature.

- EnableClassNameMapping = true
- ClassNameMappingFile = DCR-Class-Name-Mapping.xml

Setting EnableClassNameMapping property to true enables this feature. ClassNameMappingFile property specifies the XML file where the mapping between internal classnames and external (displayable) class names are specified.

This XML file should be present in the “WEB-INF/classes” directory. The applications need to make sure that all classnames are entered properly in the XML mapping file.

The following is the DTD for the classname mapping file:

```
<?xml version="1.0"?>
```

**CISCO CONFIDENTIAL**

```

<!DOCTYPE Mappings [
<!ELEMENT Mappings (ClassNameMap+)>
<!ELEMENT ClassNameMap (InternalClass, ExternalClass)>
<!ELEMENT InternalClass (#PCDATA) >
<!ELEMENT ExternalClass (#PCDATA) >
]>

```

The following is the sample classname mapping file:

```

<?xml version="1.0"?>

<!DOCTYPE Mappings [
<!ELEMENT Mappings (ClassNameMap+)>
<!ELEMENT ClassNameMap (InternalClass, ExternalClass)>
<!ELEMENT InternalClass (#PCDATA) >
<!ELEMENT ExternalClass (#PCDATA) >
]>

<Mappings>
 <ClassNameMap>
 <InternalClass>:DFM:VASA:DFMObject</InternalClass>
 <ExternalClass>DFMObject</ExternalClass>
 </ClassNameMap>
 <ClassNameMap>
 <InternalClass>:DFM:VASA:DFMObject:AccessPort</InternalClass>
 <ExternalClass>AccessPort</ExternalClass>
 </ClassNameMap>
 <ClassNameMap>
 <InternalClass>:DFM:VASA:DFMObject:Device</InternalClass>
 <ExternalClass>Device</ExternalClass>
 </ClassNameMap>
 <ClassNameMap>
 <InternalClass>:DFM:VASA:DFMObject:Device:Cable</InternalClass>
 <ExternalClass>Cable</ExternalClass>
 </ClassNameMap>
 <ClassNameMap>
 <InternalClass>:DFM:VASA:DFMObject:Device:ContentNetworking</InternalClass>
 <ExternalClass>ContentNetworking</ExternalClass>
 </ClassNameMap>
</Mappings>

```

The following is the enhancement to Object Grouping Server (OGS)1.4.2 release:

**Enhancements to evaluateGroup API**

The evaluateGroup method is an overloaded API in the OGSInterface. The evaluateGroup() API now takes an additional argument, taglist, which is a String array. This String array returns the values of the tag lists passed, for each of the group.

This enhanced evaluateGroup() method can be used when the caller wants to evaluate the group to display the devices and subgroups, and to retrieve the group properties as tag-value pairs.

The API signature is as follows:

```

public OGSObjectList evaluateGroup(OGSSecurityContext context,
 String task,
 OGSRuleContainer filter,
 String groupName,
 String[] attributeList,
 Integer membershipFormat,
 Boolean recomputeMembers,
 String[] tagList) throws OGSException;

```



CISCO CONFIDENTIAL

## CHAPTER 31

# Using the Common Services Transport Mechanism

---

The Common Services Transport Mechanism (CSTM) provides a single, consistent, simple and platform-agnostic method for handling all types of programmatic communications, including:

- Inter-Process Communication (IPC): Multiple processes on the same machine.
- Remote Procedure Calls (RPC): Multiple processes on different machines.
- In-Process Calls: Processes executing under the same virtual machine.

The following topics describe CSTM and how to use it in your applications:

- [Understanding CSTM](#)
- [Installing CSTM](#)
- [Controlling CSTM Logging](#)
- [Publishing Objects](#)
- [Accessing Published Objects](#)
- [Handling Special Requirements](#)
- [Using the CTMTest Tools and Samples](#)
- [Guidelines for Using CSTM](#)

For more information about CSTM, see

- The CSTM Functional Specification, ENG-124878.
- CSTM: Software Unit Design Specification: ENG-155861.
- CSTM User guide: ENG 161448.

## Understanding CSTM

In the past, the way to create rich application-to-application communication has been to employ DCOM, CORBA, or RMI. These rich environments typically require that applications use the same object model at both ends of a connection, which is often impractical. They also assume that both sender and the receiver have full knowledge of the message context and do not encode meta-information. This gives good performance, but makes it hard for intermediaries to process messages. Finally, each system uses a different binary encoding, making it hard to build systems that interoperate.

**CISCO CONFIDENTIAL**

CSTM provides a way to handle communications by abstracting them, providing the same API for all of them, and founding the API on well-know non-proprietary standards.

In CSTM, communications across machines are handled using XML and Serialized Java Objects over HTTP transport. Communications on the same machine are handled with Sockets. During inter-thread communication, CSTM plays a part during initial reference resolutions only.

Even though CSTM encapsulates the use of XML for message transfer, it also offers compatibility with a well-defined XML message protocol (SOAP). Use of XML RPC facilitates means there is no need to care about what operating system, programming language or object model is being used on either the client or server side.

## Installing CSTM

CSTM supports a wide variety of communications involving one or more machines. Depending on how you plan to implement it, you will want to install it using the procedures in one of the following sections:

- [Installing Basic CSTM, page 31-2](#)
- [Installing CSTM with the Tomcat Servlet Engine, page 31-3](#)

## Installing Basic CSTM

This is the basic installation for a single machine without a Java servlet engine. To install CSTM on a single machine:

- 
- Step 1** Before installing CSTM:
- If you have a previous version of CSTM installed, delete any old `ctmregistry` and `ctmregistry.backup` files. These are normally found in the same directory as the `CTM.jar` file.
  - Ensure that you have JDK 1.3.1 or later installed.
  - CSTM is supplied on the CWCS SDK disk as a WAR file. If needed, you can download CTM 1.0 and 1.1. CTM 1.0 is available at the following URL:  
[http://wwwin-nmbu/auto/cw/cdimages/ctm1\\_0/daily/NT\\_CTM1\\_0\\_INTEGRATION\\_READY/kits/](http://wwwin-nmbu/auto/cw/cdimages/ctm1_0/daily/NT_CTM1_0_INTEGRATION_READY/kits/). CTM 1.1 is available at  
[http://wwwin-nmbu/auto/cw/cdimages/ctm1\\_1/daily/NT\\_CTM1\\_1\\_INTEGRATION\\_READY/kits/](http://wwwin-nmbu/auto/cw/cdimages/ctm1_1/daily/NT_CTM1_1_INTEGRATION_READY/kits/). The same CSTM WAR file can be used for both Solaris and Windows. Extract the WAR file into a suitable location.

**Step 2** Add `CTM.jar` and `log4j.jar` to the classpath.

**Step 3** Save the `ctm_static_registry.txt` and `ctm_config.txt` files in the same directory as the `CTM.jar` file.



**Note** As soon as CSTM begins running, it will create `ctmregistry` and `ctmregistry.backup` files in the same directory as the `CTM.jar` file. Do not tamper with these files while running CSTM.

**Step 4** To run the CSTM samples, add `samples.jar` to the classpath. The source code for all samples is in `samples_source.jar`, which is available from the CSTM portal at <http://embu-web.cisco.com/eng/teams/Mjollnir/> (follow the CTM or CSTM link under “Subprojects”).

---



**CISCO CONFIDENTIAL**

## Installing CSTM with the Tomcat Servlet Engine

For full functionality, you will want to install CSTM on one or more machines with a servlet engine. CWCS supports the Tomcat servlet engine by default. No other servlet engine is supplied with CWCS, and the CWCS team recommends that you use Tomcat exclusively.

To install CSTM with a servlet engine, perform the basic CSTM install on each machine (using the instructions in the “[Installing Basic CSTM](#)” section on page 31-2), then ensure that:

- The servlet engine is installed and running on each machine. This is necessary to handle remote calls.
- You have updated the CTMServlet parameter in the CSTM configuration file (ctm\_config.txt) with the URL for the servlet engine you have installed on that machine.

**Note**

To make CSTM work with any servlet engine, you *must* modify the `CTM_URL` parameter in the `ctm_config.txt` file to point to the appropriate CTMServlet URL.

To install CSTM using the Tomcat servlet engine, follow the steps below.

- Step 1** Make sure you have Tomcat 3.2.1 or higher installed.
- Step 2** In the Tomcat installation’s conf directory, in the `server.xml` file, under the heading “Special webapps” and after the Context path for examples, add the following lines: f

```
<Context path="/ctm"
docBase="drive:/ctm"
debug="1"
reloadable="true">
</Context>
```

(Where `drive:/ctm` is the directory where CSTM is currently installed).

- Step 3** In your CSTM directory, create a WEB-INF directory.
- Step 4** Copy the `web.xml` file in the CSTM distribution into the WEB-INF directory.

**Note**

The `web.xml` file has entries to start off TestServlet, which exposes the class TestClass using the Unique Resource Name CheckServlet. This TestServlet class file is included in `samples.jar`.

- Step 5** In the WEB-INF directory, create a lib directory and place `CTM.jar` in it. If your application needs any other jars, place them in this directory.

## Controlling CSTM Logging

CSTM uses log4j to handle log messages. You can adjust CSTM log4j logging as explained in the following topics:

- [Setting Up CSTM Logging](#)
- [Viewing the CSTM Log File](#)

**CISCO CONFIDENTIAL****Note**

CSTM uses log4j for logging, but does not set up or initialize the log4j framework during runtime. If your product already ships with a log4j configuration file, you must either add CSTM-related log4j categories to it or add them to log4j.properties and add the location of log4j.properties to your application's classpath. All entries in the log4j.properties file that ships with CSTM are commented out by default.

## Setting Up CSTM Logging

By default, a running instance of CSTM on a single machine logs all messages of FATAL severity to the console. If you have a running log4j server, it will log these messages to the ctm.log in the log4j server directory.

To set up log4j logging and ensure that it works as desired, you must:

- In log4j.properties: Set the logging levels as needed.
- In log4j.properties: Set the log destination you want.
- Include the log4j.jar and log4j.properties file directory in the classpath, and start a log4j server.

The following topics explain how to perform these tasks:

- [Setting the CSTM Logging Levels](#)
- [Changing the CSTM Logging Destination](#)
- [Starting a Log4j Server](#)

**Note**

CSTM does not explicitly load the log4j.properties file. If your product is already using log4j, you may not need to have a separate log4j.properties file for CSTM. Instead, you can add the CSTM logging levels and appenders to your existing log4j properties file.

## Setting the CSTM Logging Levels

Log4j message logging levels are set using `log4j.category` entries in the log4j.properties file.

The default logging level is FATAL, and is set by the `log4j.rootCategory=FATAL` entry. The other logging levels are DEBUG, INFO, WARN and ERROR.

You can change the level of logging for CSTM or any of the CSTM modules using `log4j.category` entries in the log4j.properties file. For example: You could set explicit category entries for individual CSTM modules as follows:

```
log4j.category.CTM.server=DEBUG
log4j.category.CTM.client=INFO
log4j.category.CTM.registry=WARN
```

All CSTM modules inherit and use the `log4j.rootCategory` setting unless there is a `log4j.category` set for them.

## Changing the CSTM Logging Destination

In log4j, server message destinations are called appenders. Appenders are created using `log4j.appender` entries in the log4j.properties file. Any appender can be assigned to a logging levels using the logging level's `log4j.category` entries in the same file.

## CISCO CONFIDENTIAL

In the default `log4j.properties` file, only the console appender is defined, and the `log4j.rootCategory` that sets the default logging level is set to the console appender. If you are running a `log4j` server, all messages will be sent first to the server, then to the `ctm.log` file (if this `log4j` server is not running, the messages will only be logged to the console).

It is possible to define appenders for your own purposes. For example: On a single machine, you may have multiple virtual machines making CSTM calls. To have all CSTM log messages coming from all the VMs logged to a single `ctm.log` file, you would update the `log4j.properties` file with necessary appenders to send the messages to the `log4j` server and then run a `log4j` server. Every VM will then send its log messages to the `log4j` server, which in turn logs it into the `ctm` file.

To define and use appenders, you must:

- Update the `log4j.properties` file with appenders to which you want messages to be sent.
- Run the `log4j` server with the new appenders.

For example: To create an appender named “A2”, add the following socket appender configurations to the `log4j.properties` file:

```
log4j.appender.A2=org.apache.log4j.net.SocketAppender
log4j.appender.A2.RemoteHost=localhost
log4j.appender.A2.Port=8888
```

Then, to have your custom logging levels all send messages to this appender, you would update your `log4j.category` entries as follows:

```
log4j.category.CTM.server=DEBUG, A2
log4j.category.CTM.client=INFO, A2
log4j.category.CTM.registry=WARN, A2
```

## Starting a Log4j Server

Once categories and appenders are defined, you can start the `log4j` server by adding `log4j.jar` to the classpath and then issuing the following command:

```
>> org.apache.log4j.net.SocketServer port proppath propdir
```

Where:

- `port` is the port on which the server is to run.
- `path` is the complete path and file name for the `log4j.server.properties` file.
- `dir` is the directory in which `log4j.server.properties` file is stored.

For example: if `log4j.server.properties` is stored in `D:\ctm`, and the `log4j.jar` and `log4j.properties` files are also stored there, you would start the server with the following commands:

```
>> set classpath = d:\ctm\log4j.jar;d:\ctm
>> org.apache.log4j.net.SocketServer 8888 d:\ctm\log4j.server.properties d:\ctm
```



### Note

The `ctm.log` file will be created in the same directory from which this command was executed.

## Viewing the CSTM Log File

You can view the `ctm.log` file with any ASCII text editor. For each message, the log will include the:

- Date: The message date in `dd/mm/yyyy` format

**CISCO CONFIDENTIAL**

- Time: The message time in hh:mm:ss:msec format.
- Logging level: INFO, DEBUG, WARN, ERROR, or FATAL
- Classname: The complete name of the class that output the message.
- Methodname: The method name of the calling method.
- Message: The text of the message logged from the calling method.

For example:

```
31/Jan/2002 17:52:31:984 DEBUG com.cisco.nm.xml.ctm.server.CTMServer <clinit> -
Serverport:40000 MinThreads:3 maxThreads:20
```

For message sizes larger than 32MB, you must set the startup memory to a higher value. To do this, compile using the javac option for startup memory. For example, compile using `>> Javac -J-Xms48m` to set startup memory to 48MB, or use `>> Javac -J-Xms64m` to set it to 64MB.

## Publishing Objects

When you publish an object, you expose it to other processes. The CSTM files and CSTM Server API provide several means for publishing and unpublishing objects. The following topics explain these methods and the guidelines you should follow when:

- [Publishing Objects Statically, page 31-6](#)
- [Publishing Objects Dynamically, page 31-7](#)
- [Handling Remote Objects, page 31-7](#)
- [Publishing Objects Securely, page 31-8](#)
- [Unpublishing Objects, page 31-9](#)

## Publishing Objects Statically

CSTM allows you to register classes statically by adding them to the file `ctm_static_registry.txt`. Entries in the static registry file take the form:

```
urn=classname,
```

where:

- `urn` is the Unique Resource Name of the object
- `classname` is the object's classname.

Static registration using `ctm_static_registry.txt` always takes precedence over dynamic registration done using a Publish call (see [“Publishing Objects Dynamically” section on page 31-7](#)). For example: If `ctm_static_registry.txt` contains the entry `abc=TestClass`, and you later try to publish a resource with the URN `abc` and classname `TestClass`, CSTM will report that this resource is already registered. If a class is registered statically, then CSTM will read and use the static registry entry whenever your client invokes that object.

To use static registration successfully, always ensure that:

- The `ctm_static_registry.txt` file is stored in the same directory as `CTM.jar`.
- All classnames appearing in `ctm_static_registry.txt` also appear in the `Classpath`.

## CISCO CONFIDENTIAL

# Publishing Objects Dynamically

CSTM provides methods to publish objects dynamically, at runtime. Objects published dynamically are available via all the types of communication CSTM supports, including In-Process Calls, IPC, and RPC.

Ideally, the published object should expose its functionality via an interface. If the exposed object provides an interface, the CSTM client can invoke the exposed object either by binding to this interface using `CTMClientProxy`, or as a generic `CTMClient`. If the exposed object does not provide an interface, you can use only `CTMClient`. Note that the `CTMClientProxy` feature is available only if you are using JDK 1.3.1 or later.

To publish an object dynamically, use the `CTMServer.publish` method:

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(java.lang.String urn, java.lang.Object
object, com.cisco.nm.xml.ctm.common.CTMServerProperties properties) throws CTMException
```

`CTMServer.publish` takes the following input arguments:

<i>urn</i>	A Unique Resource Name for the object you want to publish. URNs need only be unique for the management application area or server on which they are used.
<i>object</i>	The published object's classname.
<i>properties</i>	Additional properties for the published object.

For example: To publish a single reference of class `TestClass`, with the URN `xyz` (in this case the same object will be used, irrespective of the type of client; this functionality is not available with static registration):

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(xyz, new TestClass())
```

To publish the same class with the same URN, by passing the class definition (in this case the functionality varies based upon the type of client):

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(xyz, TestClass.class)
```

## Handling Remote Objects

`CTMServer` provides methods to register and publish remote objects, as follows:

- [Registering Remote Objects Statically](#)
- [Registering Remote Objects Dynamically](#)
- [Publishing Remote Objects](#)

## Registering Remote Objects Statically

Static registration can be done by adding an entry in the static CSTM registry file (`ctm_static_registry.txt`).

```
Urn name = class name
```

**CISCO CONFIDENTIAL****Example**

```
abc=com.cisco.cmf.test.TestServer
```

Since this kind of registration is done at compile time, it does not need any API.

**Registering Remote Objects Dynamically**

The CTMServer class provides methods to publish the object dynamically at runtime. The published object is available through all types of communication such as external applications (Non-Appliance), Inter-Appliance, Intra-Appliance and Intra-JVM. If there are errors, CTMServer throws CTMException.

If the exposed object implements an interface, it can be used by the CSTM Client as a remote interface. CSTM client can invoke the exposed object by binding to this interface using CTMClientProxy or by using generic CTMClient, which does not use this remote interface. But generally, it is recommended to implement the interface since it allows the CSTM clients to use CTMClientProxy.

**Usage**

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(
 java.lang.String urn,
 java.lang.Object object,
 com.cisco.nm.xml.ctm.common.CTMServerProperties properties)
```

where urn is any string, including spaces or even an empty string as "".

**Publishing Remote Objects**

You can publish a remote object in two ways:

- Passing a single reference

To publish class `TestClass`, with URN name `xyz`, using the same object regardless of type of client:

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(xyz,
 new TestClass())
```

- Passing the class definition

To publish class `TestClass` with URN `xyz`, by passing the class definition.

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(xyz,
 TestClass.class)
```

In this case, the functionality varies based upon type of client.

**Publishing Objects Securely**

CSTM allows you to publish objects securely. CSTM uses CWCS security to secure the access to the published object.

The CTMServerProperties is passed in the Publish call to set the security options. This means that any remote object published securely with a set of user authorized roles, can only be accessed by an authorized user, whose role/permissions are then authenticated against the ones set while publishing.

When the CTMServerProperties is not given as a parameter to the publish method, then by default, the security option is treated as false, that is, no security checks are performed on the clients attempting to access this resource.

## CISCO CONFIDENTIAL

The security option makes most sense in the context of calls coming from outside the box. In such cases, the publisher might intend to secure the access to the published resource. Hence, security checks are performed only for the calls coming from outside the box. Security checks are not performed for the calls within the same box.

However, in the `CTMServerProperties` option, whenever the security option is set to true, it must have a list of roles that are authorized to access this URN.

The `CTMServerProperties` for a published URN (Unique Resource Name) can be specified with a specific security value, that is, security value of True or False depending on whether security needs to be turned ON or OFF. Also, if security is turned On, the string [] of roles gives the list of roles that are authorized to securely access this URN. When no arguments are provided to the `CTMServerProperties` call, then the default security value is set to "False" and the string [] of roles is set to null.

### Usage

```
public CTMServerProperties()
public CTMServerProperties(boolean security,
String [] roles)
```

### Input Arguments

*security* A boolean value that indicates if security is true/false

*roles* A string array that contains the names of roles authorized to access the published resource/URN

### Example

To publish a class called `TestClass`, with security options turned on:

1. Set the permissions allowable/authorized roles

```
String [] roles = {"CsAuthServlet.SA",
"CSAuthServlet.NA"};
```

2. Set these permissions in the `CTMServerProperties`

```
com.cisco.nm.xml.ctm.common.CTMServerProperties props =
new com.cisco.nm.xml.ctm.common.CTMServerProperties(true,
roles);
```

3. Publish `TestClass`, with security options

```
com.cisco.nm.xml.ctm.server.CTMServer.publish(xyz,
TestClass.class,
props);
```

To publish this URN with security option turned off:

```
com.cisco.nm.xml.ctm.server.publish("abc",
TestClass.class)
```

## Unpublishing Objects

Unpublishing an object cancels exposure of that object and its functionality via CSTM. Note that, when you unpublish a remote object, you can re-use its URN.

**CISCO CONFIDENTIAL****Note**

If your application publishes a URN through CTM Server, make sure you unpublish the URN when your application no longer requires it to be maintained or exposed. If your application uses CWCS, this means you must (at a minimum) unpublish the URN during a Daemon Manager “stop” notification.

To unpublish an object, use the `CTMServer.unpublish` method:

```
com.cisco.nm.xms.ctm.server.CTMServer.unpublish(java.lang.String.urn) Throws CTMException
```

`CTMServer.unpublish` has only one input argument, `urn`, which is the Unique Resource Name of the object to be unpublished.

For example:

To unpublish the URN `xyz`:

```
com.cisco.nm.xms.ctm.server.CTMServer.unpublish(xyz)
```

## Accessing Published Objects

CSTM provides three ways to invoke the server side functionality from a client:

- `CTMClient`: Uses the URN of the remote object and a static method of a class which takes the method signature. For details, see the [“Using CTMClient” section on page 31-10](#).
- `CTMClientProxy`: Obtains a proxy for the corresponding remote object. For details, see the [“Using CTMClientProxy” section on page 31-11](#).
- `CTMCall`: Instantiates a call that maintains a connection with the remote object. `CTMCall` provides static methods for binding and invoking remote object methods. For details, see the [“Using CTMCall” section on page 31-13](#).

Developers should choose the method that best suits their application’s needs.

All three of these methods are affected by or make use of:

- `CTMServer.properties`, which sets variables that control remote sessions. For details, see the [“Changing CTM Client Properties” section on page 31-14](#).
- `CTMConstants`, which sets the encoding style for client sessions. For details, see the [“Using CTMConstants” section on page 31-15](#).
- The `ctm_config.txt` CSTM configuration files, which provide additional parameters. For details, see the [“Using the CTM Configuration File” section on page 31-15](#).
- `CTMException`, which is the generic exception handler. For details, see the [“Handling CTM Exceptions” section on page 31-17](#).

## Using CTMClient

`CTMClient` provides static methods for invoking a remote object, and supports user- defined exceptions. If the server-side application throws an exception in the remote method, `CTMClient` re-throws the same exception. CTM will throw `CTMException` whenever there is an error during publishing or invoking a remote object.

For IPC and RPC communications, you can provide the remote host IP address as a parameter to the `CTMClient.invoke` call. The data is then tunneled over HTTP.



**CISCO CONFIDENTIAL****Usage**

```
com.cisco.nm.xml.ctm.client.CTMClient.invoke(
 java.lang.String urn,
 java.lang.String host,
 java.lang.String methodName,
 java.lang.Object[] parameters
 CTMClientProperties properties,
 CTMSerializer returnType,
 CTMParameterDesc[] par)
 throws Exception
```

```
com.cisco.nm.xml.ctm.client.CTMClient.invoke(
 java.lang.String urn,
 java.lang.String host,
 java.lang.String methodName,
 java.lang.Object[] parameters
 CTMClientProperties properties,
 CTMSerializer returnType,
 throws Exception
```

**Input Arguments**

<i>urn</i>	The Unique Resource Name of the published object you want to call.
<i>host</i>	The host name or IP address of the remote host.
<i>methodName</i>	The name of the remote method that the client call wants to execute.
<i>parameters</i>	An object array of the parameters that must be passed to the remote method the client call wants to execute.
<i>properties</i>	The CTMClient properties that control the session. See the <a href="#">“Changing CTM Client Properties”</a> section on page 31-14.
<i>par</i>	Parameter descriptor that has more information about the parameter passed. This is a wrapper class for type org.apache.axis.description.ParameterDesc.
<i>returnType</i>	Argument to register or unregister the serializer or deserializer in the IMarshal interface.

**Example**

The remote object `TestClass`, with method `testmethod`, is already published with URN `xyz`. The method parameters (paralist) can be accessed as an Object [], as follows:

```
com.cisco.nm.xml.ctm.client.CTMClient.invoke("xyz",
 "testmethod",
 paralist)
```

**Using CTMClientProxy**

A client using `CTMClientProxy` has to know the interface implemented by the remote object. This means you can use `CTMClientProxy` only if the server-side object has implemented a remote interface. `CSTM` does not pose any restriction on this interface. By generating dynamic proxies, `CTMClientProxy` eliminates the need to generate stubs or a skeleton for each and every server-side remote object.

**CISCO CONFIDENTIAL****Note**


---

The CTMClientProxy feature is available only if you are using JDK 1.3.1 or later.

---

If you do not have access to a remote interface on the client side, use CTMClient (see the [“Using CTMClient”](#) section on page 31-10) instead of CTMClientProxy.

CTMClientProxy simplifies remote method invocation and hides the mechanism for passing method names and parameters to the called object. However, this does not mean that the server maintains a separate instance for every proxy object. The total number of remote objects instantiated in the server process depends on the server policy. CSTM clients should not rely on the state of a remote object.

Like CTMClient, CTMClientProxy supports user-defined exceptions. If the server-side application throws an exception in the remote method, CTMClientProxy re-throws the same exception. CSTM will throw CTMException whenever there is an error during publishing or invoking a remote object.

As with CTMClient IPC and RPC communications, you can provide the remote host IP address as a parameter to the CTMClient.invoke call. The data is then tunneled over HTTP.

**Usage**

```
com.cisco.nm.xml.ctm.client.CTMClientProxy.getProxy(
 java.lang.Class[] interfaceClasses,
 java.lang.String urn,
 java.lang.String host,
 CTMClientProperties properties)
throws CTMException
```

**Input Arguments**

<i>interfaceClasses</i>	An array containing the name of the remote- object interface(s) (a class or classes).
<i>urn</i>	The Unique Resource Name of the published object you want to call.
<i>host</i>	The host name or IP address of the remote host.
<i>properties</i>	The CTMClient properties that control the session. See the <a href="#">“Changing CTM Client Properties”</a> section on page 31-14.

Using the remote interface creates a local instance. You can then access any method of the remote interface; just use this local instance (you can also do this with an array of interfaces).

**Example**

To invoke a method called testmethod, using knowledge of the remote interface TestInterface:

```
TestInterface iTest=(TestInterface)CTMClientProxy.getProxy(TestInterface.class,
URN);
int result1 = iTest.testmethod(new byte[messageSize],
"2",
"0");
```

**CISCO CONFIDENTIAL****Using CTMCall**

CTMCall provides static methods for binding and invoking remote-object methods.

A client using CTMCall need not know the interface implemented by the remote object, eliminating the need to implement interfaces like this. Like CTMClient, the only contract required between the client and the server is the remote object's URN.

Unlike CTMClient, CTMCall maintains a socket connection with the remote object, so the same connection can be used for multiple requests. The CTMCall object will keep the remote connection alive until the object either explicitly closes the connection or the object itself is garbage-collected. Despite this, you should not rely on the state of the remote object. CTMCall will try to reconnect to the server once before throwing an exception.

Like CTMClientProxy, CTMCall is performance efficient. Note that CTMClientProxy internally constructs an instance of a CTMCall object to perform the remote invocation.

Like CTMClient and CTMClientProxy, CTMCall supports user-defined exceptions. If the server-side application throws an exception in the remote method, CTMCall re-throws the same exception. A CTMException is thrown by CTM incase of error during publishing and invoking the remote object.

As with CTMClient and CTMClientProxy, you can provide the remote host IP address as a parameter to the CTMClient.invoke call. The data is then tunneled over HTTP.

**Usage**

```
CTMCall ctmCall = new com.cisco.nm.xml.ctm.client.CTMCall(
 java.lang.String urn,
 java.lang.String host,
 CTMClientProperties properties)

ctmCall.setMethodAndInvoke(String methodName, Object parameters[])

ctmCall.setMethodAndInvoke(String methodName, Object parameters[],CTMSerializer
returnType, CTMParameterDesc[] par)throws Exception (where ENCODING_STYLE is set to soap)
```

**Input Arguments**

- urn*            The Unique Resource Name of the published object you want to call.
- host*            The host name or IP address of the remote host.
- properties*    The CTMClient properties that control the session. See the [“Changing CTM Client Properties”](#) section on page 31-14.
- par*             Parameter descriptor that has more information about the parameter passed. This is a wrapper class for type org.apache.axis.description.ParameterDesc.
- returnType*    Argument to register or unregister the serializer or deserializer in the IMarshal interface.

**CISCO CONFIDENTIAL****Examples**

To invoke `testmethod`, with parameter object `[] paraList`, when this method is defined in a class called `TestClass`, which has been exposed under the URN of `xyz`:

```
CTMCall call = new CTMCall ("xyz");
call.setMethod("testmethod", paraList);
call.invoke();
call.closeConnection();
```

To invoke an overloaded `testmethod`, with parameters object `[] paraList`:

```
CTMSerializer ctmSerializer
= new CTMSerializer(namespace, localPart, beanName),
ParameterDesc paramDesc
= new CTMParameterDesc(xmlQname, mode, xmlTypeQname, javaType)
```

When this method is defined in a class called `TestClass`, which has been exposed under the URN of `xyz`:

```
CTMCall call = new CTMCall ("xyz");
call.setMethod("testmethod", paraList, ctmSerializer, paramDesc);
call.invoke();
call.closeConnection();
```

**Changing CTM Client Properties**

Pass `CTMClientProperties` to `CTMClient`, `CTMClientProxy` or `CTMCall` in order to alter default values for the following properties required to access the remote server:

- Timeout
- Encoding Style
- CTMServlet or other remote server URL
- Access Port

The relevant constants for these properties are defined in the `CTMConstants` (see the [“Using CTMConstants”](#) section on page 31-15).

**Usage**

```
public CTMClientProperties(
int timeout,
int encodingStyle,
String url,
int port)

public CTMClientProperties(int timeout)

public CTMClientProperties(int encodingStyle)
```

**Input Arguments**

*timeout* The time (in milliseconds) that the client call waits for a response from the server before reporting a timeout exception. The default is 60000 milliseconds (1 minute). The timeout feature is currently disabled, since a connection pool is used internally.

*encodingStyle* The encoding for messages to and from the remote object. The default is BINARY.

**CISCO CONFIDENTIAL**

*url* The URL string needed to access the CTMServlet or other remote server that will handle the CTM client requests. The default value is blank.

*port* The default value for the HTTP port is 80.

**Example**

To set CTMClient Call to use encoding style CTM\_SOAP:

```
CTMClientProperties properties = new CTMClientProperties(CTM_SOAP);
```

To set the CTMClient call timeout to four minutes

```
CTMClientProperties properties = new CTMClientProperties(240000);
```

**Note**

The timeout feature is disabled since a connection pool is used internally.

## Using CTMConstants

CTMConstants is a placeholder for all the public constants defined in CSTM. Currently, the only constants defined are:

- **CTM\_BINARY**: The default encoding style for all IPC and RPC communications. Request and response objects are serialized between client and server.
- **CTM\_SOAP**: The default encoding style for third-party applications. SOAP request and SOAP response are used for communication between the client and the server

## Using the CTM Configuration File

The CSTM configuration file, `ctm_config.txt`, is stored in the same directory as the `CTM.jar` file, and sets parameters for the CSTM sessions. The configuration file entries and their default values are shown in [Table 31-1](#).

To change any of the default settings in this file, first stop all the VMs using CSTM, and then delete the `ctmregistry` and `ctmregistry.backup` files. These files are located in the same directory as the jar files.

**Table 31-1** Configuration File Parameters

Parameter	Default Value	Description
SERVER_PORT=	40000	Sets the starting port number. Ports will be used in ascending order from this port. You can customize this port.
MAX_VM_PORTS =	20	Sets the maximum number of VMs (each instance of a CSTM Server runs in a separate VM). This parameter also sets the number of ports required for CSTM (MAX_VM_PORTS + 1). The registry server will use one port, while CSTM Servers running in different VMs will use the other ports.

**CISCO CONFIDENTIAL****Table 31-1 Configuration File Parameters**

Parameter	Default Value	Description
SERVER_TIMEOUT=	7200000	Sets the time (in milliseconds; the default is 2 hours) after which the server deletes unused client entries and connection details. Note: This feature is currently disabled, per customer requirements.
MIN_THREADS=	10	Sets the minimum number of threads in the ThreadPool.
MAX_THREADS=	100	Sets the maximum number of threads in the ThreadPool.
CTM_URL=	:8080/ctm/CTMServlet	Stores the remote URL at which the CTMServlet can be accessed. The URL varies with the type of servlet engine running on the remote machine. For Tomcat, the parameter is usually set as follows: CTM_URL=:8080/ctm/CTMServlet.
CTM_SSL=	0	When set to 1, turns on SSL for URL connections. This provides secure communications between products.
THREAD_SLEEP=	180000	Sets the frequency (in milliseconds; the default is once every three minutes) at which the client connection pool-cleanup process occurs (that is, the cleanup thread will activate at this frequency).
SOCKET_IDLETIME=	600000	Sets the time (in milliseconds; the default is 10 minutes) limit after which an idle socket connection in the client connection pool is removed (that is, the socket is closed).
MAX_VM_CLIENT_CONNECTION=	10	The maximum number of connections maintained in the client connection pool per VM. Above this limit, the client will wait for a free connection. A client connection that is in use is made available when the client releases it. If this parameter is assigned a value of zero, CSTM will default it to 1 automatically.
CTM_FILE_UPLOAD_URL=	:8080/ctm/FileUpload	Specifies the remote URL at which the FileUpload servlet can be accessed. This varies with the servlet engine being run. For Tomcat it is usually the default value.
CTM_FILE_DOWNLOAD_URL=	:8080/ctm/CTMServlet FileDownload	Specifies the remote URL at which the FileDownload servlet can be accessed. This varies with the servlet engine being run. For Tomcat it is usually the default value.

**CISCO CONFIDENTIAL****Table 31-1 Configuration File Parameters**

Parameter	Default Value	Description
REGISTRY_LOCATION=	D:/Progra~1/CSCOpX/MDC/tomcat/webapps/cwhp/WEB-INF/lib	Sets the location of a common registry to be used under Tomcat. You must specify this value if you want to use a common registry. By default, the registry location is set to the directory where ctM.jar was installed.
REGISTRY_SERVER_TIME_OUT=	20000	Sets the socket timeout (in milliseconds) for reading from the common registry server. You should configure this value as appropriate for the number of concurrent publish actions you expect for the application.
REGISTRY_SERVER_RETRIES	3	Sets the number of times CSTM should attempt to connect to the common registry server.
SECURITY_WRAPPER	com.cisco.nm.xml.ctm.security.CMFSecurityWrapper	Identifies the class name of the security implementation used for authentication and authorization of remote calls through the servlet. This class must implement the com.cisco.nm.xml.ctm.common.ISecurityWrapper interface.

## Handling CTM Exceptions

During publishing and invoking of remote objects, CSTM throws the generic CTMException. CTMException is thrown only by CSTM. [Table 31-2](#) lists the CTMExceptions and their occurrence conditions.

CTMException also provides messages indicating when an exception occurred, as shown in [Table 31-3](#). You can retrieve the invocation status using the following call:

```
int getInvocationStatus();
```

To ensure that exceptions received by CSTM from the underlying environment are wrapped in CTMException, use the following API call:

```
Throwable getException(); // returns the wrapped exception
```

With Binary encoding, CSTM also supports user-defined exceptions. Individual applications can throw their own exceptions on exposed methods of a remote object, and CSTM will re-throw the same exception on the client side.

In SOAP encoding, CSTM always throws CTMException. SOAP does not support user-defined exceptions. To make the remote exception information available, use a `getMessage()` call.

**Table 31-2 CTMException Messages and Conditions**

Message	Occurs When
DUPLICATE_URN	The URN is already registered.
URN_NOT_FOUND	Accessing a URN not present in the registry.

**CISCO CONFIDENTIAL****Table 31-2** CTMException Messages and Conditions (continued)

Message	Occurs When
ERR_IN_OBJECT_CREATION	The exposed class definition is in the Classpath during compile time, allowing the code to compile, but is not visible later. Can also occur if the exposed class does not have a zero argument constructor.
ERR_IN_INVOCATION	An incorrect method signature is provided. Can only occur when using CTMClient or CTMCall. When using CTMClientProxy, an incorrect method signature will prevent the code from being compiled.
ERR_IN_CLIENT_CREATION	The client event cannot be created.
ERR_IN_CTM_SERVER_STARTUP	The CSTM server cannot start up because the registry is corrupt or cannot be created (that is, the directory containing the jar files is not writable). Occurs on server startup only.
ERR_IN_REGISTRY	An error is noticed in the registry, either because the registry has been removed or corrupted. Can also occur when either the server or the client are attempting to access the absent or corrupt registry.
REQ_ID_INVALID	The client request REQ_ID is invalid, corrupt, or has timed out. The server associates every request with an internal REQ_ID, which it times out after the interval specified in the SERVER_TIMEOUT parameter.
UNAUTHORISED_ACCESS	The user was trying to access a remote method whose URN was published with security options on, and the user could not be authorized.
USER_NOT_AUTHENTICATED	The user does not have the required permissions or role to access the remote object or method. When publishing a remote method, the publisher can specify if the particular URN needs to be accessed securely and register a set of roles that are authorized to use it.
ENCODING_STYLE_NOT_SUPPORTED	There is an attempt to use an encoding style other than Binary or SOAP.
ERR_IN_SOAP_MARSHAL_METHOD_ARGS	The CSTM client cannot convert the method name and arguments into a SOAP request message. This error can occur due to the argument objects not being serialized, failure to register the soap serializer, or other SOAP errors.
ERR_IN_SOAP_UNMARSHAL_METHOD_ARGS	The CSTM server cannot unmarshal the method name and arguments from the SOAP request message. This error can occur due to the argument objects not being serialized back to java objects, failure to register the soap serializer, or other SOAP errors.
ERR_IN_SOAP_MARSHAL_RESULT	The CSTM server cannot marshal the result into a SOAP response message. This error can occur due to failure to register the soap serializer or other SOAP errors.
ERR_IN_SOAP_UNMARSHAL_RESULT	The CSTM client cannot convert the SOAP response back to java objects. This error can occur due to the serializer for the returned object not being registered with CSTM or other SOAP errors.
ERR_IN_SOAP_SERIALIZER_REGISTER	An attempt to register the SOAP serializer fails.
ERR_IN_SOAP_SERIALIZER_UNREGISTER	An attempt to unregister the SOAP serializer fails.



**CISCO CONFIDENTIAL****Table 31-2** *CTMException Messages and Conditions (continued)*

Message	Occurs When
ERR_IN_CONNECTING_TO_SERVER	Trying to connect to the server for the first time (that is, opening a connection).
ERR_IN_WRITING_REQUEST	The client's attempt to write the request fails. This is usually due to an error in the connection or the object stream.
ERR_IN_READING_REQUEST	The server's attempt to read the client request fails.
ERR_IN_WRITING_RESPONSE	The server's attempt to write the response fails.
ERR_IN_READING_RESPONSE	The client's attempt to read the response from the server fails.
ERR_IN_RELEASING_CONNECTION	Attempting to release the connection between a client and the server.
UNKNOWN_EXCEPTION	Error conditions or exceptions not covered by the other conditions in this table. CTMException will catch conditions caused by general Java exceptions and send the exception stack trace information with the UNKNOWN_EXCEPTION comment.

**Table 31-3** *CTMException Invocation Status Messages*

Message	Indicates
BEFORE_INVOCATION	Exception occurred before invoking the method on the server.
AFTER_INVOCATION	Exception occurred after invoking the method on the server.
UNKNOWN_INVOCATION_STATUS	CSTM cannot determine whether the method was invoked on the published object.

## Handling Special Requirements

The following topics provide guidelines for using CSTM to handle special requirements, including:

- [Implementing Secure CSTM Clients](#)
- [Running Registry Server as a Separate Process](#)
- [Registering the CSTM Port](#)
- [Using SOAP Encoding With CSTM](#)
- [Using the IMarshal Interface](#)
- [Using IMarshal's Register Method](#)
- [Performing CSTM File Transfers](#)
- [Retrieving HTTP Errors](#)

### Implementing Secure CSTM Clients

CSTM exposes published objects securely with the help of the underlying security service. The CWCS framework provides interfaces for validation of the request (authentication/authorization). Security is only applied for RPC communication, that is, if the request originates from a CSTM client on the same machine, CSTM server will allow access any remote object on that box.

**CISCO CONFIDENTIAL**

Registry handler provides list of user roles that can access a particular URN and the underlying security service does the actual validation.

Regardless of encoding style, CSTM uses HTTP transport for RPC communication. So CSTM servlet acts as front-end and receives all incoming requests. The servlet locates the corresponding CSTM server with registry client interface and delegates the request. Applications can also publish objects in servlet engine VM. Before forwarding the request, the servlet will validate (authentication/authorization) the request if URN is exposed securely.

If security on a remote machine is turned on, the CSTM client calls (CTMClient, CTMClientProxy and CTMCall) must perform authentication before calling a published object on that remote machine.

CSTM uses the CWCS shared-secret mechanism to authenticate client access. In the shared-secret scheme, each CSTM client is logged on as a user, and thus has a user name and password. The administrator maps each user name to a particular role, and each user also registers a password, called a shared secret, with CWCS. In order for the CWCS security server to authenticate this user, the client must provide the shared secret and his login information in the HTTP stream. This login and shared secret information is coded in a cookie tag in the HTTP stream.

The CWCS security server, for each securely published URN, first receives the HTTP stream from the client and checks for the cookie tag in the stream. It then does authentication checks for this user against the published resource and its allowable roles.

The following is an example of a CTMCall using security options:

```
import com.cisco.nm.cmf.security.secret.SecretClient;
import com.cisco.nm.xms.ctm.common.CTMClientProperties;
public class TestSecureCall
{
 public static void main(String arg[])
 {
 String username = "admin";
 String secret = "admin";
 //host where the resource is published.
 String host = "sujathab-nt";
 //published URN(unique resource name)
 String urn = "abc";

 SecretClient sc = new SecretClient();
 String httphost = "http://" + host + ":1741";
 String cookie = sc.secretLogon(httphost,
 username,
 secret);
 CTMClientProperties Properties =
 new CTMClientProperties();
 Properties.setHttpHeaderEntry("Cookie", cookie);
 }
}
```

The CTMClientProperties is sent out as part of the CTMClient call in CTMClient, CTMCall or CTMClientProperties. Refer to TestSecureCall.java and TestSecureClientProxy.java, in the samples.jar file.

Note that, if a call to a published object on the local machine is made that explicitly refers to the host with the IP address of the local machine or host name of the local machine, the call will be treated as remote call and security checks will be performed on it. Security checks are not performed for any other local calls, where the host is not specified or the host value is "localhost".

## CISCO CONFIDENTIAL

### Running Registry Server as a Separate Process

Registry Server need not share the process space of other processes. We recommend that you start RegistryServer as a separate process, since this will reduce the overhead required to start Registry Server on demand (during synchronous call execution).

You can use the `isRegistryServerRunning()` API to identify whether the Registry Server has started. Your application must do this for any application processes that depends on Registry Server to start.

### Registering the CSTM Port

If your application uses CSTM ports, you need to register the Registry Server port in the `/etc/services` file. Your application should do this during the application install, and remove the port during uninstall.

To register the port properly, your application installation should do the following:

1. Know the package in which `ctm.jar` is shipped
2. Check that the server port is free. This should be done during the installation prerequisites check. If the port is not free, it should select a random port (and check whether it is already in use as well).
3. Update this port number in the `/etc/services` file
4. Update the `SERVER_PORT` value in the `ctm_config.txt` file. This should be done during the post-install procedure.
5. Mark this file `ctm_config.txt` as volatile.

We also recommend that you register the port with CANA.

To ensure uniformity across applications using CWCS, follow the port naming convention `cscxxxxcstm`, where `xxx` is the name of your application. For example: For Common Services, the CSTM Registry Server port is named `cscocscstm`.

### Using SOAP Encoding With CSTM

You can invoke server side functionality with SOAP encoding using either `CTMClient` or `CTMCall`. For details on the differences between these two types of invocations, see the [“Using CTMClient” section on page 31-10](#) and the [“Using CTMCall” section on page 31-13](#).

In both cases, you must have the client set the encoding property in `CTMClient.properties` to `CTM_SOAP` (see the [“Changing CTM Client Properties” section on page 31-14](#)). Once this property is set, the invocation process on the client side is basically the same as the process you use when encoding is set to the default `CTM_BINARY`.

However, if any of the arguments of the invoked method is a user-defined class, you must call `IMarshal`'s `register` function first, before invocation (see the [“Using IMarshal’s Register Method” section on page 31-24](#)).

`IMarshal` registration is needed to serialize all the parameters into equivalent SOAP representations. Generally speaking, you will need to register whenever:

- You want to send a SOAP request containing instances of one or more user-defined classes to a CSTM server for processing.
- The CSTM server will return an instance of a user-defined class as a result of your method invocation.

**CISCO CONFIDENTIAL**

- You have a CSTM client that wants to create a SOAP request. The client will need to register all the classes which are going to be a part of the request.

For example: You have a client which would like to invoke a method on the server. One of the method's arguments is an instance of a user-defined class, or composite object. For this argument to be serialized into an equivalent SOAP representation, CSTM requires that the client register this composite object with IMarshal. Once this is done, CSTM can generate an equivalent SOAP query for the input arguments of the method to be invoked.

The requirement is the same when an attempt is made to deserialize an incoming SOAP request containing an instance of a composite object. You must pre-register every incoming user-defined class with IMarshal.

Another example: You have a client which would like to invoke a function of a server application. CSTM accepts the parameters needed to invoke this function, `methodName`, parameters for the method, `args` and `urn` that identifies the server/resource whose functionality is to be invoked. CSTM invokes the function from the IMarshal interface and passes the input from the client.

The call then moves to CTMSoapMarshaller, which uses Apache Axis to create a SOAP-XML query based on the input from the client. This query (SOAP-XML) is then returned by the interface to CSTM, which transports it to the server. After the processing in the server, a SOAP-XML response is generated, which is transported by CSTM to the client. This response is converted to an equivalent Java Object using IMarshal interface and then passed to the client.

Consider another scenario, in which you have an external application, which would like to invoke a functionality of a server API. The application supports XML-SOAP compatible interface, that is, the interface generates SOAP queries and accepts SOAP responses. A typical example would be third parties or external applications, which may use CSTM to invoke functionality within the network management solution and pass a SOAP-XML query as input.

The SOAP-XML query message would contain the required data to invoke the method as well as the URN to identify the service. This query message is fed to CSTM. CSTM passes this query to a function supported by IMarshal, which uses Apache Axis to extract the data contained in the query (the name of the method to be invoked, a set of arguments for it, and the URN that identifies the service to be invoked). CSTM uses this data to invoke the service. CSTM converts the response from the service to an equivalent SOAP response using Apache-Axis and this is then returned by CSTM to the external application.

## Using the IMarshal Interface

The IMarshal interface, which is used to access the functionality of Apache Axis, is as follows:

```
package com.cisco.nm.xml.ctm.common;
public interface IMarshal
{
 public String marshalMethodAndArgs(String urn, String methodName, Object args[]) throws
 Exception ;

 public RPCData unmarshalMethodAndArgs(byte[] message) throws Exception ;

 public byte[] marshalReturnValue(String urn, String methodName, Object returnValue) throws
 Exception ;

 public Object unmarshalReturnValue(byte[] message) throws Exception ;

 public Object unmarshalReturnValue(byte[] message, CTMSerializer
 returnType,CTMParameterDesc[] paraDesc)

 public void register(CTMSerializer serializer) throws CTMException;
```

**CISCO CONFIDENTIAL**

```
public void unregister(CTMSerializer serialzer) throws CTMException;
}
```

## Using marshalMethodAndArgs

This function is used to generate an equivalent SOAP-XML query for an input.

The method uses the following Apache-Axis classes:

RPCElement	Stores data that forms the body of a SOAP envelope. The attributes of the object of this class are the name of the method which was invoked, the urn of the service and the arguments/return value of the method.
SOAPEnvelope	Defines a SOAP envelope and has methods to extract the data contained in it.
ServiceDescription	Indicates the type of message that is being created (a request or response) and the type of encoding desired.
Message	Defines methods used to convert a SOAP envelope into a string or byte array. Acts as a placeholder for SOAP-XML messages.

### Input parameters

methodName	Method name to be invoked
urn	Unique identity for a specific service
args'	Arguments needed to invoke the method

### Output

Equivalent SOAP-XML query based on the input parameters.

## Using unmarshalMethodAndArgs

This method is used to deserialize a SOAP-XML message (the byte array input) to extract the following information: name of the method to be invoked, the arguments to be supplied to the method and the URN that identifies the service.

This method returns a user-defined class called RPCData. The attributes of this class are the name of the method to be invoked, the service urn and the arguments for the method.

### Input parameters

A byte array that contains the SOAP\_XML message to be deserialized.

### Output

An object of type RPCData with the following attributes:

**CISCO CONFIDENTIAL**

<code>urn</code>	Unique resource name that identifies the service to be invoked
<code>methodName</code>	Name of the method to be invoked
<code>arguments</code>	Arguments to the method to be invoked

**Using unmarshalReturnValue**

This method is used to extract the return value (Java object) of the method that was invoked.

**Input parameters**

- A byte array, which contains a SOAP-XML message which needs to be deserialized.
- A byte array, which contains a SOAP-XML message which needs to be deserialized, the return type of the CTMSerializer, and the parameter description.

**Output**

Return value (Java object) of the method that was invoked.

**Using marshalReturnValue**

This method generates a SOAP-XML message containing the result of the method.

**Input parameters**

<code>methodName</code>	The name of the method that was invoked
<code>urn</code>	The urn identifying the service
<code>object</code>	The result of the method

**Output**

A byte array containing a SOAP-XML message representing the input parameters.

**Using IMarshal's Register Method**

This method is intended for use in situations where you want to use SOAP encoding and one or more of the arguments of the remote method is a user-defined class (see [“Using SOAP Encoding With CSTM” section on page 31-21](#)). The signature of the register method in the IMarshal interface is described below:

```
public void register(CTMSerializer serializer) throws CTMException;
public void register(CTMSerializer serializer, String dataType) throws CTMException;
```

**Input Arguments**

The input argument *serializer* wraps the required registration information shown below.

**CISCO CONFIDENTIAL**

namespace	The namespace for the composite object.
localPart	The local name for the composite object or the (xsi:type)
BeanName	The name of the class which defines the composite object. The path location of the class must be included in the name.

**Usage**

```
public CTMSerializer(
String namespace,
String localPart,
String beanName)
```

The implementation of the IMarshal interface is in CTMSoapMarshaller.java.

**Example**

We want to invoke `testMethod`, which has `testObject` as one of its arguments. In the client code where we invoke `CSTM`, we would do the following:

1. Set the encoding in `CTMClientProperties` to “CTM\_SOAP”.
2. Register the composite object which needs to be serialized, as follows:

```
Class _cls=null;
_cls = Class.forName("com.cisco.nm.xml.ctm.soap.
CTMSoapMarshaller");
Method m = _cls.getMethod("getCTMSoapMarshaller",null);
IMarshal soap = null;
soap =(IMarshal) m.invoke(soap , null);
```

3. With the instance of `IMarshal`, call the register method and pass `CTMSerializer`, which will wrap the namespace, localname and the name of the class which contains the definition of the object:

```
soap.register(new CTMSerializer(
namespace,
localPart,
beanName));
```

4. Invoke `testMethod` as with any other client.

The namespace in which a user-defined class is registered in `CSTM` client should be consistent throughout. For more examples of the use of `CSTM` and `SOAP` encoding, refer to the following files in `samples-source.jar`: `TestSoapCall.java`, `TestSoapClient.java`.

**Performing CSTM File Transfers**

You can perform file uploads and downloads to and from a remote machine using the `CTM File Transfer` utility.

`CTMFileTransfer` provides separate methods for upload and download. Both methods throw `CTMFileTransferException`, which is explained in the [“About CTMFileTransferException”](#) section on page 31-27.

`CTMFileTransfer` can be used when the application requests a file download or a file upload.

**CISCO CONFIDENTIAL**

Consider the scenario when an application requests a file download from another machine. In this case, the call from the download method of CTMFileTransfer is directed to the FileDownload servlet. The servlet is invoked as a web server and downloads the required file.

The case for FileUpload is similar to the one given for file download except that in this case the upload method of CTMfileTransfer is called which invokes the FileUpload servlet.

**CTMFileTransfer Client Side Functionality**

The two methods exposed by CTMFileTransfer are upload and download.

```
public static void download(String destFile , String srcIP , String srcFile)throws
CTMFileTransferException ;
public static void upload(String srcFile , String destIP , String destFile)throws
CTMFileTransferException ;
```

**Using CTMFileTransfer Upload**

This method is invoked to upload a file to a remote machine/box. The input arguments to the method are:

srcFile	Name of source file to be uploaded
destFile	Name of destination file
destIP	Destination IP address to which the file must be uploaded

The method calls the FileUpload servlet and begins to upload the file in chunks of 1024 bytes at a time. This is done to reduce the amount of memory consumed by the program for transfer of large files.

A URLConnection is established to the FileUpload servlet using the destination IP address. If the connection is not established a CTMFileTransferException.Host\_Unreachable is thrown. The name of the destination file and data read from the source file is transmitted to the servlet.

If the source or destination files are not found, the following exceptions are thrown:

- CTMFileTransferException.Source\_File\_Not\_Found
- CTMFileTransferException.Destination\_File\_Not\_Found

After the transfer is completed, if the size of the source file and the destination file is found to be unequal then a CTMFileTransferException.Transfer\_Interrupted is thrown.

**Using CTMFileTransfer Download**

This method is invoked to download a file from a remote machine/box. The input arguments to the method are:

srcFile	Name of the source file to be downloaded.
destFile	Name of the destination file
destIP	Destination IP address from which the file must be downloaded



## **CISCO CONFIDENTIAL**

The method calls FileDownload servlet and begins to download the file in chunks of 1024 bytes at a time. This is done to reduce the amount of memory consumed by the program for transfer of large files.

A URLConnection is established to the FileDownload servlet using the destination IP address. If the connection is not established a CTMFileTransferException.Host\_Unreachable is thrown. The name of the source file is transmitted to the servlet, the servlet then transmit the data read from the source file to the client.

If the source or destination files are not found, the following exceptions are thrown

- CTMFileTransferException.Source\_File\_Not\_Found
- CTMFileTransferException.Destination\_File\_Not\_Found

After the transfer is completed, if the size of the source file and the destination file is found to be unequal then a CTMFileTransferException.Transfer\_Interrupted is thrown

### **CTMFileTransfer Server Side Functionality**

The two servlets in the CSTM server are FileUpload and FileDownload.

#### **FileUpload Servlet**

The FileUpload servlet is invoked by the upload method of CTMFileTransfer. It receives the name of the destination file to which data has to be written. After verifying the existence of the file, the servlet writes data sent by the client to this file. If the destination file is not present, an exception is thrown and the client is notified about it.

#### **FileDownload Servlet**

The FileDownload servlet is invoked by the download method of CTMFileTransfer. It receives the name of the source file from which data is to be read. After verifying the existence of the file, the servlet sends it to the client. If the source file is not present, an exception is thrown and the client is notified about it.

### **About CTMFileTransferException**

The following CTMFileTransferException messages are thrown for various conditions.

<b>Message</b>	<b>Cause</b>
SOURCE_FILE_NOT_FOUND	Incorrect source file name or the source file is not found.
DESTINATION_FILE_NOT_FOUND	Incorrect destination file name or destination file name not found.
HOST_UNREACHABLE	The specified host cannot be reached. This is usually due to an invalid IP address or the fact that no web server is running on the specified IP address.
TRANSFER_INTERRUPTED	The file transfer was interrupted
DESTINATION_INVALID	The destination is invalid.

**CISCO CONFIDENTIAL**

## Retrieving HTTP Errors

You can get HTTP error codes and response messages using `CTMException` and one of the following three calls.

```
Hashtable h = ctMex.getHttpErrorHash();
getHttpStatusCode()
getHttpResponseMessage()
```

Note that in the `getHttpErrorHash()` call, the hashtable contains the error code and error messages, with the keys `ErrorCode` and `ErrorMessage`, respectively.

Which call you use depends on the JRE you are using, and the value you want. If you are using JRE 1.3.1\_04, use the `getHttpErrorHash()` call only. If you are using JRE 1.3.1\_06 or later, use any of these three calls and get the required value (code, message, or both).

Note that only the `getHttpErrorHash()` call will have both the Error Message and ErrorCode values, and both will be converted into strings.

## Using the CTMTest Tools and Samples

You can use the CTMTest tool to:

- Get a feel for the operations that CSTM lets you perform.
- Check the exception cases and error cases and make sure they are relevant.
- Check the performance using multiple scenarios.

You can execute the CTMTest operations using the test class in the `samples.jar` file. The test file lets you:

- Set the message size of an arbitrary byte stream.
- Invoke a client using short command line calls: CL for `CTMClient`, CLP for `CTMClientProxy`, or CLL for `CTMCall`.
- Set the timeout value as needed.
- Simulate the processing delay in the server method.

While running CTMTest, you can use the up-arrow and down-arrow to access previous and next commands you issued. Depending on your platform and command history, the CTMTest command history is stored across invocations. For example, if you access CTMTest and then exit, the next time you access CTMTest, you should still be able to access the history from the previous invocations. This is handy when you are trying to perform `CTMClient` calls with multiple options.

While calculating the number of messages per second, CTMTest tracks delays in accessing the serialized parameter file and does not use them in the calculation. Calculated time taken for each of the Client operations does not include parameter-file access delays.

The following topics describe several useful applications of the CTMTest tool, including:

- [Creating a Custom Test File, page 31-29](#)
- [Publishing a Test Object, page 31-29](#)
- [Unpublishing a Test Object, page 31-29](#)
- [Accessing a Test Method Using CTMClient, page 31-30](#)
- [Accessing a Test Method Using CTMClientProxy, page 31-30](#)

## CISCO CONFIDENTIAL

- [Accessing a Test Method Using CTMCall](#), page 31-31
- [Testing CSTM Communications](#), page 31-31

### Creating a Custom Test File

In order to test with your own classes rather than the test classes given in the samples.jar, you must:

- Add your new class to the classpath.
- Serialize the parameters that need to be passed in to your class.

You can do this by writing your own file, and including the serialization methods given in the Serializer class. The example file MyTestClass.java shows how to add the parameters to an object array and serialize them into a file.

If the parameters that need to be passed to the method are composite objects, they will need to implement Serializable and will need to have a no-argument constructor. The name of this serialized file will then be given as a command line argument to the CTMTest tool when you run the client CL, CLL or CLP commands.

### Publishing a Test Object

To publish a test object:

- 
- Step 1** At the command prompt, enter `java CTMTest`. The CTMTest tool runs.
- Step 2** At the `CTMTEST>>` prompt, enter the command `P URN Classname Option`, where:
- `URN` is the Universal Resource Name you want to assign to the object.
  - `Classname` is the name of the class to be exposed.
  - `Option` is `-s` or blank if you want to publish by passing a single reference to the class, or `-d` to publish by passing the class definition.
- 

### Unpublishing a Test Object

To unpublish a test object:

- 
- Step 1** At the command prompt, enter `java CTMTest`. The CTMTest tool runs.
- Step 2** At the `CTMTEST>>` prompt, enter the command `U URN` where `URN` is the Universal Resource Name assigned to the previously published object.
-

**CISCO CONFIDENTIAL****Accessing a Test Method Using CTMClient**

To access a method on a test object using CTMClient:

---

**Step 1** At the command prompt, enter `java CTMTest`. The CTMTest tool runs.

**Step 2** At the `CTMTEST>>` prompt, enter the following command:

```
CL URN methodname parmlist options
```

Where:

- *URN* is the Universal Resource Name of the object whose method you want to call.
  - *methodname* is the name of the method you want to call.
  - *parmlist* is the name of a file containing the serialized parameters for this method. If this file is not in the current directory, specify the filename with entire path.
  - *options* is one or more of the following:
    - `-c` is the number of clients to spawn. The default is 1. If you choose to spawn multiple clients, the system starts off separate threads for each CTMClient request.
    - `-i` is the IP\_address or server name of the remote host to be accessed. The default is `localhost`.
    - `-t` is the CTMClient Timeout value, in milliseconds. The default is 5000. This is useful for setting the timeout value of the CTMClient call when it has delays accessing the server.
    - `-n` is the number of messages to be sent. The default is 1000. Changing this value is useful when you know you will have many messages to be sent.
- 

**Accessing a Test Method Using CTMClientProxy**

To access a method on a test object using CTMClientProxy:

---

**Step 1** At the command prompt, enter `java CTMTest`. The CTMTest tool runs.

**Step 2** At the `CTMTEST>>` prompt, enter the following command:

```
CLP URN methodname parmlist options
```

Where:

- *URN* is the Universal Resource Name of the object whose method you want to call.
- *methodname* is the name of the method you want to call.
- *parmlist* is the name of a file containing the serialized parameters for this method. If this file is not in the current directory, specify the filename with entire path.
- *options* is one or more of the following:
  - `-c` is the number of clients to spawn. The default is 1.
  - `-i` is the IP\_address or server name of the remote host to be accessed. The default is `localhost`.
  - `-t` is the CTMClient Timeout value, in milliseconds. The default is 5000. This is useful for setting the timeout value of the CTMClient call when it has delays accessing the server.

## CISCO CONFIDENTIAL

- *-m* is the message size. The default is 1000 bytes. Changing this value is useful if you know that the specific test method has a larger message size defined in the TestInterface.
- *-n* is the number of messages to be sent. The default is 1000. Changing this value is useful when you know you will have many messages to be sent.

## Accessing a Test Method Using CTMCall

To access a method on a test object using CTMClientProxy:

**Step 1** At the command prompt, enter `java CTMTest`. The CTMTest tool runs.

**Step 2** At the `CTMTEST>>` prompt, enter the following command:

```
CLL URN methodname parmlist options
```

Where:

- *URN* is the Universal Resource Name of the object whose method you want to call.
- *methodname* is the name of the method you want to call.
- *parmlist* is the name of a file containing the serialized parameters for this method. If this file is not in the current directory, specify the filename with entire path.
- *options* is one or more of the following:
  - *-c* is the number of clients to spawn. The default is 1.
  - *-i* is the IP\_address or server name of the remote host to be accessed. The default is localhost.
  - *-t* is the CTMCall Timeout value, in milliseconds. The default is 5000. This is useful for setting the timeout value of the CTMCall when it has delays accessing the server.
  - *-n* is the number of messages to be sent. The default is 1000. Changing this value is useful when you know you will have many messages to be sent.

## Testing CSTM Communications

If you want to test CSTM communication between applications on the same machine:

1. Open two CTMTest windows.
2. Send messages from one window to the other.
3. Try to connect to a published resource in the first command window from the second window.
4. Try to publish and perform client access from the same command window.

To simulate CSTM communications between applications on different machines:

1. Make sure Tomcat 3.2.1 is running on the machine that will act as the server.
2. Make sure that CSTM is registered with Tomcat on the server by following the steps in the [“Installing CSTM with the Tomcat Servlet Engine”](#) section on page 31-3.

**CISCO CONFIDENTIAL**

3. Publish the class on the server.
4. Access the exposed resource from one or more clients by specifying the IP address of the server machine.

## Using the Sample TestClass

The samples file `samples.jar` contains a sample class called `TestClass`, which:

- Includes `TestMethod`.
- Includes the local variable `Count`.
- Implements `TestInterface`.

`TestMethod` does the following:

- Maintains the `Count` variable.
- Increments `Count` by 2 every time it is invoked.
- Makes a byte array of a size you can specify.
- Allows you to test timeout parameters by setting a sleep time value as a parameter.
- Contains a demonstration composite class object as a parameter. Please note that for composite classes, you must implement the interface `Serializable` and also have a zero-argument constructor.

To set these parameters for `TestMethod`, you set the values into the parameter array and serialize it into a file with a specific name. For example, you might enter `>>java MyTestClass 60000000 0 bigfile`, where:

- `60000000` is the size of the byte array (message size). In this case, the message size is 60Mbytes.
- `0` is the simulated delay (in milliseconds) in the method.
- `bigfile` is the name of the serialized file.

This serialized file can then be given as input to the `CTMTest` command prompt, so that the respective parameters are set in the particular method. The class `MyTestClass` is also included in the `samples.jar` file.

## Using the CSTM Samples

The `CSTM samples.jar` file contains the `CTMTest` utility java files and classes, plus several other test and sample java files and classes. [Table 31-4](#) shows the java and class files included in `samples.jar`.

You can use these sample files and classes with `CTMTest` to perform a variety of useful tests, including:

- [Testing Parameter Passing, page 31-33](#)
- [Testing for Timeout Errors, page 31-33](#)
- [Testing Multiple Clients, page 31-34](#)

**CISCO CONFIDENTIAL****Table 31-4** Java and Class Files in Samples.jar

File/Class Name	Description
CTMTest	The CTMTest utility main class.
TestClass	This is accessed from the CTMTest utility. This class includes TestMethod and implements the TestInterface.
TestInterface	This Interface class is implemented by TestClass and is used by CTMClientProxy.
Serializer	Use methods of this class to serialize the method parameters.
MyTestClass	An instance of a class using Serializer methods to serialize the parameters to the TestMethod defined in TestClass.
CompositeObject	A file used to demonstrate that the parameters passed to the method of an exposed class can be a composite object. An instance of a composite object is passed to the TestMethod defined in TestClass.
TestServer	Standalone test programs for testing TestServer alone.
TestClient	Standalone test program for testing TestClient alone.
TestClientProxy	Standalone test program for testing TestClientProxy alone.
TestCall	Stand alone test program for testing TestCall alone.
TestServlet	Standalone test program for testing publishing an object from a servlet.

## Testing Parameter Passing

You must first make a serialized file to pass the parameters to the test method:

```
>> java MyTestClass 1000 0 1kfile
```

Then start the CTMTest tool:

```
>> java CTMTEST
```

Publish the object containing the test method

```
CTMTEST>> P xyz TestClass
```

You can then pass the serialized file to the published method, using any of the three CSTM client calls:

```
CTMTEST>> CL xyz testmethod 1kfile -n 1000
CTMTEST>> CLL xyz testmethod 1kfile -n 1000
CTMTEST>> CLP xyz testmethod -n 1000 -m 1000
```

In the case of the CTMClientProxy call (CLP), remember that the remote interface is hard coded, so it always uses TestInterface and TestMethod. However, the message size parameter can be varied, so there is no need to use the serialized file to pass parameters in this case, because the other parameters are hardcoded.

## Testing for Timeout Errors

Use the following call to create a serialized file with message size set to 1000bytes and the server side delay set to 5000 milliseconds:

```
>> java MyTestClass 1000 5000 1k_5msec_file
```

**CISCO CONFIDENTIAL**

Then publish the TestClass:

```
CTMTEST>> P xyz TestClass
```

Then make a client call with the timeout set to 1000 milliseconds:

```
CTMTEST>>CL xyz testmethod 1k_5msec_file -n 1000 -t 1000
```

This will cause timeout error conditions, since the server delay is set at 5000 milliseconds, but the client timeout is set at 1000 milliseconds. The client will wait for 1000 milliseconds to hear from the server, and once the server calls a sleep method for 5000 milliseconds, the client call will timeout.

**Testing Multiple Clients**

Use the -c option to test for multiple (three, in this case) CTMClient calls:

```
CTMTEST>> CL xyz testmethod 1kfile -c 3 -n 1000
```

**Guidelines for Using CSTM**

Following are the recommendations or guidelines that applications should follow when using CSTM:

- Starting Registry Server

It is recommended that you start CSTM Registry Server as a separate process, since it receives most of the CSTM Calls (both server and client). Starting Registry Server in a heavily loaded environment like Tomcat is also not advisable.

Sample code is as follows:

```
import com.cisco.nm.xms.ctm.registry.*;
import org.apache.log4j.Category;
public class StartCTMRegistryServer
import com.cisco.nm.xms.ctm.registry.*;
import org.apache.log4j.Category;
public class StartCTMRegistryServer
{
 static Category cat = Category.getInstance("CTM.Registry");
 public static void main(String arg[])
 {
 if(!CTMRegistryServer.isRegistryServerRunning())
 {
 CTMRegistryServer.startRegistryServer();
 cat.debug("Registry Server started successfully");
 }
 else
 {
 cat.debug(" Registry Server was started successfully in other
JVM");
 }
 }
}
```

- Proper usage of CSTM Client APIs provided by CSTM
  - For applications which have both CSTMServer and CSTMClient on the same machine:



## CISCO CONFIDENTIAL

- If the requirement is to have several calls from the client to the server in a short span of time, create an instance CTMCall or CTMClientProxy object and use the same object for multiple calls, finally call CloseConnection. This reduces unnecessary overhead of creating sockets on the client side.
- If calls to the server are made infrequently, CTMClient can be considered.
  - For applications which have CSTMServer and CSTMClient on different machines:  
Any instance of CTMCall or CTMClient or CTMClientProxy can be used.

Multithreaded-environment related guidelines:

- On Client side:  
If the application wants to share same instance of CSTM client object, from multiple-threads, then use an instance of CTMClientProxy class, which is thread-safe and synchronized.
- On Server side:
  - Application can expose CSTM urn for either reference to the object whose functionality needs to be exposed or its class.
  - Application should supply the reference, if exposed object is thread-safe. In this case CTM will use the same object over simultaneous or multiple incoming requests and also Client should rely on the state of the server object.
  - Instead of reference if Class is supplied, CTM creates separate instance of remote object. This object gets garbage collection, after client loses connection with the server by either explicit close or when client call object gets garbage collected.
- CSTM also wraps HTTPReponse error code with CSTM Exception that occurs on the client side.  
When CTMException is encountered on CTM client side, application can also check `ctmexception.getHttpResponseMessage` to get message / `ctmexception.getHttpResponseCode` for Code or `ctmexception.getHttpErrorHash` to get both error code and message as key value pairs.
- Application should properly unpublish all the Urns that they publish on processes shutdown. It is required for CSTM to properly cleanup CTM registry. When the processes start up later, this will avoid any port-in-use issues.
- The CSTM configuration file, `ctm_config.txt`, is stored in the same directory as the `CTM.jar` file, and sets parameters for the CSTM sessions. Applications can change these default settings depending on their needs. See [Using the CTM Configuration File](#) for more details.

***CISCO CONFIDENTIAL***



CISCO CONFIDENTIAL

## CHAPTER 32

# Using Package Support Updater

---

The Package Support Updater (PSU) helps your application check for software and device support updates, download them to the server file system along with the related dependent packages, and install them. PSU provides the user with a central location from within the CiscoWorks application, to all software updates and device packages for better management of their network.

PSU makes use of configuration of CCO credentials and Proxy Server settings from a CWCS Web Server page. Command line tools can perform installation, unistallation and download of device packages. You can also download software updates using CLI.

Software updater includes applications using the IDUs model for delivery of software updates and device support, Common Services Service Packs, point patches, etc.

You can navigate to the software update related user interfaces from the Common Services Application. Select Software Center > Software updates from Common Services to see the software updates.

Software updates allows you to:

- View a list of solution bundles installed along with a list of products and versions installed.
- View a list of updates installed (base version, patch versions, service pack versions).
- Select one or more or all products and request for a 'Check for updates' function, which lists all available updates, from which you can choose the required updates and download it to a location to install.
- Configure to look Cisco.com to check for packages available for updates.
- Download the selected updates to any location on the CiscoWorks server, from where, you can later install those packages using existing package install procedures [for example, extracting the package, and running the *setup.sh*].

The following topics describe how to use PSU with your application:

- [Understanding PSU](#)
- [Using PSU with Your Application](#)
- [Working with Software Center](#)

For more details on the PSU dialogs, see the *Software And Device Updates Software Functional Specification*, EDCS-310850.

[http://wwwin-eng.cisco.com/Eng/ENM/CMF/CMF2\\_3/PSUSW.doc](http://wwwin-eng.cisco.com/Eng/ENM/CMF/CMF2_3/PSUSW.doc)

**CISCO CONFIDENTIAL**

## Understanding PSU

PSU provides software package updates by downloading them from a designated source to CiscoWorks Server. PSU is designed to download and install updates for device packages like CiscoView device packages, Synchronous Device Interface (SDI) and application data packages like CMIC adapters package, which contain device specific data.

PSU allows you to view a list of solution bundles installed, along with products list and versions. You can also get a detailed list of all OS level packages with their version numbers.

## Using PSU with Your Application

PSU helps you to check for software and device support updates, download them to your server along with the related dependent packages. You can then install them on your server.

For software updates, PSU help you to look for updates from Cisco.com, and download them to your server location. You can then install the updates using the procedures recommended in the readme file of the update package.

**Note**

---

No web based install tool will be provided for installing software updates due to technical limitations.

---

For device support updates, PSU helps you to look for updates from Cisco.com and download them to your server location. You can then install the updates using a web based user interfaces wherever possible. You can also install directly from Cisco.com

Most of the Common Incremental Device Support (CIDS) based packages can be installed directly from CiscoWorks Homepage. PSU will not support installation of classic IDU monolithic packages. These have to be installed manually by the user with the help of the Readme files.

You can also choose to install the selected packages directly from Cisco.com without saving them on your server.

Following sections describe how to use PSU:

- [Integrating Applications](#)
- [Integrating Applications](#)
- [Backing Up the Server](#)
- [Releasing Package Updates](#)
- [Uninstalling Device Support Packages](#)

## Integrating Applications

Applications should follow the guidelines mentioned below to use PSU:

- Add New tags in the INFO file
- Register with PSU
- Implement Package Adapter and Package Descriptor Interfaces

**CISCO CONFIDENTIAL****Adding New Tags in INFO Files**

Applications should use the tags defined by install framework in the info files to specify the applications and packages.

To identify your product among the list of all installed products, the corresponding INFO file should have the following tags:

```
PROD_INFO=<Name of the INFO file>
```

```
PRODNAME=<Name of the Product>
```

For example, to identify the product updates from RME, the INFO file rme.info should be updated with the following tags:

```
PROD_INFO=rme
```

```
PRODNAME=Resource Manager Essentials
```

**Registering with PSU**

You should register your application during install time by invoking PSURegistration API. This API registers your application with PSU by creating appropriate directories and updating PSU configuration files.

**Syntax:**

On Solaris:

```
PSURegistration "Product_Name" "Adapter_directory_location" "Shortname:Longname",
"extraClasspaths"
```

On Windows:

```
PSURegistration ("Product_Name" "Adapter_directory_location" "Shortname:Longname",
"extraClasspaths")
```

**Table 32-1 Arguments and their Description**

Arguments	Description
Product_Name	Short Name of the application. Examples: rme, cmf
Adapter_directory_location	Location where the adapter zip files are stored. There will not be any adapter implementation for products which does not support device updates
Shortname	Tag used by the CCO script to query the updates. The tags defined by the product are as follows: <ul style="list-style-type: none"> <li>• cmf (CiscoWorks Common Services)</li> <li>• rme (Resource Manager Essentials)</li> <li>• cvw (CiscoView)</li> <li>• nmim (Integration Utility)</li> <li>• dfm (Device Fault Manager)</li> </ul>

**CISCO CONFIDENTIAL****Table 32-1 Arguments and their Description**

Arguments	Description
Longname	Name of the product. Usually this refers to the Marketing name of the product and should match the PRODDNAME tags defined in the info files.
extraClassPaths	Any other classes other than the standard directories. This is required by the PSU CLI script to add them to the java classpath before execution

**Examples:**

On Solaris:

```
PSURegistration "rme" "/auto/cw/cdimages/rme4_0_blr/image/disk1/rme/adapter"
"rme:Resource Manager Essentials"
```

On Windows:

```
PSURegistration("cmf","", "cw2000:CiscoWorks Common Services,"")
```

**Implementing Package Adapter and Package Descriptor Interfaces**

Applications should implement the following interfaces for installing the device packages and packaging them along the update image before releasing to CCO.

- com.cisco.nm.xml.psu.interfaces.adapter.PkgAdapterIf
- com.cisco.nm.xml.psu.interfaces.pkg.PkgDescrIf

**Using the PSU Command Line Tools**

You need to login to the CiscoWorks server to execute CLI. PSU provides CLI features to query all packages from user specified package directory location. Queries will also list details about packages associated with specified package if any.

You can also install all or specified package from user specified package directory location. CLI will install all packages from Source Location or install latest versions of installed packages.

In case of installing packages from Cisco.com, you have to first download the packages from CCO, save them to a directory in your computer and then install them by specifying that directory.

PSU CLI will also uninstall all installed packages or uninstall specified package.

**Package Support Updater Usage**

```
psu -help
psu -p <product1,product2...> -query -src dir {-all|PackageNames}
psu -p <product1,product2...> -install -src dir {-all|PackageNames} [-noprompt]
psu -p <product1,product2...> -uninstall {-all|PackageNames} [-noprompt]
psu -p <product1,product2...> -download -dst dir
psu -p <product1,product2...> -software -dst dir
```

From Common Services 3.0 Service Pack 2, the following are also supported:

```
psu -p <product1,product2...> -pkgDependents [-src dir] {-all|PackageNames}
psu -p <product1,product2...> -pkgVersion [-src dir] {-all|PackageNames}
```

**CISCO CONFIDENTIAL****Commands**

```

-help (-h): Print psu command usage
-query (-q): Print list of packages (default source location is installed repository of
the product).
-install (-i): Install packages (from user specified directory).
-uninstall (-u): Uninstall packages.
-software (-s): download all software packages
-product (-p): product for which packages are to be downloaded
-pkgDependents (-pdep) : Print list of base package(s) for the specified package(s)
present in the source location (default source location is installed repository of the
product).
-pkgversion (-pver) : Print the versions of the specified package(s) present
in the source location (default source location is installed repository of the product).

```

**Options**

```

-src <dir>:install packages from user specified directory for installation.
-all: select all packages available at the source location.
-noprompt: flag which turns off the prompting for restart of daemon services during
install/uninstall.
-dst: destination directory for downloading.

```

**Examples**

```

psu -p rme -q -all [Lists all the packages in the installed repository for RME]
psu -p rme -q -src <dir> [Lists all the packages in the specified directory for RME]
psu -p rme -i -src <dir> -all [installs all packages for RME from user specified
directory]
psu -p rme -u -all [uninstalls all packages of RME from the installed repository]
psu -p rme -d -all -dst<dir> [downloads all device packages for RME to user specified
directory]
psu -p rme -s [downloads all software packages for RME to the specified directory]
psucli -p rme -src /opt/psupkgs -pdep Rtr3600
psucli -p rme,cmf -src /opt/psupkgs -pver Mdf Rtr3600

```

**Note**


---

The PSU in Solaris is `<NMSROOT>/bin/PSUcli.sh`. In NT, it is `<NMSROOT>/bin/PSUcli.bat`

---

**Backing Up the Server**

To backup what is installed on the server, PSU maintains package and device map in the respective product packages directory. Package map is a list of all packages installed on the server and Device map is a list of all the supported devices on the server for that product. Any PSU configuration settings will be backed up for that product as part of CMF backup, and restored during a CMF restore.

Maintaining package map and device map is helpful in two ways.

- Whenever you want to view installed packages, PSU will not need to iterate through installed repository to look for package descriptors and associate OIDs to MDF Name.
- If there is a problem with the server, PSU will restore package map and device map to show what devices and packages were installed on the server.

**CISCO CONFIDENTIAL**

## Releasing Package Updates

Update packages may be a software package, a device package or a IDU package.

A Software package that PSU can identify is a ZIP file comprising software image and PSU descriptor. Similarly IDU package that PSU can identify is a ZIP file with IDU bundle and PSU Descriptor. And Device Package is a ZIP file with Device Package related files and PSU descriptor.

It is expected that PSU descriptor will contain all the relevant methods to facilitate PSU in recognizing package as software, IDU or device. Additional method added to existing PSU descriptor to identify it for software, device or IDU is `getType()` method.

Applications should follow the prescribed interface for defining PSU descriptor. For more details, see Appendix B in the Software And Device Updates Software Functional Specification (EDCS-310850) available at: [http://wwwin-eng.cisco.com/Eng/ENM/CMF/CMF2\\_3/PSUSW.doc](http://wwwin-eng.cisco.com/Eng/ENM/CMF/CMF2_3/PSUSW.doc)

Applications must provide a PSU header, and a PSU Metafile, along with software, device or IDU package.

The PSU Header is parsed to get the list of dependent packages, which are automatically selected and installed while installing the updates.

The PSU Meta data file, which is an XML file, is updated in CCO for every version of application, bundle, or for every device and software update for that version of the application. The PSU Meta data file contains the information of all updates (device or software) pertaining to the application. The PSU Meta data file has to be downloaded to check the availability of packages based on the list of applications or bundle posted in CCO. The PSU Meta data file improves the performance of PSU by avoiding the download of all PSU headers of all CCO updates.

## Uninstalling Device Support Packages

PSU may deny an uninstall operation based on the application's inability to handle the uninstall. You have to perform the necessary checks before uninstalling a package, through the adapter.

You can select one or more products and unistall device packages. This displays a list of packages installed with version numbers. You can then select a subset of the packages and choose to uninstall them from the server

For basic information on PSU (including autogenerated code documentation, installation procedures, packages, dependencies, and utilities supplied with it), see the “[About the Package Support Updater \(PSU\) Components](#)” section on page 29-5.

For more information about PSU, see the Software And Device Updates Software Functional Specification (EDCS-310850) available at:

[http://wwwin-eng.cisco.com/Eng/ENM/CMF/CMF2\\_3/PSUSW.doc](http://wwwin-eng.cisco.com/Eng/ENM/CMF/CMF2_3/PSUSW.doc)

## Working with Software Center

Software Center helps you to easily check for software and device support updates, download them to their server file system along with the related dependent packages, and install them.

Software Center helps you to look for software and device updates from CCO, and download them to a server location from which you can install the updates. In the case of device updates, Software Center helps you to install the updates using a web based user interfaces, wherever possible.



## CISCO CONFIDENTIAL

Most of the device family based packages can be installed directly from the web interface, while the device support packages like IDU, have to be installed based on the installation instructions documented in the respective readme files.

You may also uninstall a device support package. Software Center does not support uninstallation of software updates. In the case of software updates, only a download is allowed. For device updates of a particular application, (eg: RME), in addition to downloads, install and uninstall is also supported.

To backup what is installed on the server, Software Center maintains a package and device map in the respective applications installed packages directory. Package map is a list of all packages installed on the server and device map is a list of all the supported devices on the server.

Software Center provides a Command Line Interface to download device updates and software updates, and install or uninstall device packages.

The following sections elaborate the Software Center features:

- [Performing Software Updates](#)
- [Performing Device Updates](#)
- [Scheduling Device Downloads](#)
- [Viewing Activity Logs](#)

## Performing Software Updates

The Software Updates tab under Software Center takes you to the Installed Software dialog box. Here, you can see two tables: One that lists the Bundles installed and another that lists the products installed. The bundle or product name, the version, and the date on which the software was installed are given in the tables.

You can click on each product to view a list of updates installed such as base version, patch versions or service pack versions. You can also further drill down each product and get a detailed list of all individual OS level packages installed on the system, along with the versions.

To download all software updates for selected products:

- 
- Step 1** Configure Cisco.com credentials in **Server > Security > Cisco.com Connection Management > User setup**.
  - Step 2** Select **CommonServices > Software Center > Software Updates > Download Updates**.  
A wizard leads you through the process.
  - Step 3** Click **Next**.  
A Destination Location Screen appears.
  - Step 4** Specify **Destination Location** folder.
  - Step 5** Click **Next**.  
The summary of the products to be downloaded appears.
  - Step 6** Click **Finish** to confirm download of the available packages.  
If you click **Cancel**, the default Installation Software dialog box appears.
- 

To select software updates for selected products:

**CISCO CONFIDENTIAL**

- 
- Step 1** Configure Cisco.com credentials in **Server > Security > Cisco.com Connection Management > User setup**.
- Step 2** Select **Common Services > Software Center > Software Updates > Select Updates**.  
A wizard leads you through the process. The Available Image screen is displayed. For each Product Name you can find the Type (patch, SP4, etc.), Installed Version, Available Version, Readme Details, Posted Dates, and Size.
- Step 3** Select the product you need to update.
- Step 4** Click **Next**.  
A Destination Location Screen appears.
- Step 5** Specify **Destination Location** folder.
- Step 6** Click **Next**.  
A summary of the products selected is displayed.
- Step 7** Click **Finish** to confirm download of the selected packages.
- Step 8** If you do not want to add the selected packages, click **Back** to reselect packages or click **Cancel** to exit.
- 

## Performing Device Updates

The Software Updates link under Software Center tab takes you to the Device Update window. The default summary screen displays a count of devices supported for each product installed in the system. Click on the product name to view a list of all device support packages installed and the version of each package. You can also view a list of all support device types, by clicking on the respective device type count, against the product.

To check for updates:

- 
- Step 1** Select **Common Services > Software Center > Device Updates**  
The Device Updates page appears.
- Step 2** Select the check box corresponding to the product for which you want to check for updates, then click **Check for Updates**.  
The Source Location page is displayed. You can check for updates at Cisco.com or at a Server.
- Step 3** To check for updates at Cisco.com:  
  - a. Select the Cisco.com radio button.
 To check for update from a Server:  
  - a. Select the Enter Server Path radio button.
  - b. Enter the path or browse to the location using the **Browse** tab.



**Note** If the source location is Cisco.com, make sure the credentials are configured in the **Server > Security > Cisco.com Connection Management > User setup**.

---

- Step 4** Click **Next**.

## CISCO CONFIDENTIAL

The Available Packages and Installed Packages page is displayed.

**Step 5** Select the package name that you wish to update, then click **Next**.

The Device Update page appears. You can choose to install device packages or download device packages. To install device packages, select the **Install Device Packages** radio button. To download device packages, select the **Download Device Packages** radio button.

If you select Install Device Packages:

- Click **Next**. A summary of your inputs is displayed.
- Click **OK** to confirm. A warning message pop up to tell you that the daemons are restarted.
- Click **OK** to continue with installation.

If you select Download Device Packages, Enter the folder in File Selection field or click **Browse** to select the folder.

**Step 6** In Scheduling pane, choose the Run Type (Immediate or Once), Date, and Time. In the Job Info field, provide the Job Description and E-mail ID.

---

## Scheduling Device Downloads

You can schedule device package downloads and specify the time, frequency (daily, weekly) of the downloads. PSU uses Job Resource Manager (JRM) for scheduling download jobs.

PSU supports the following download policies:

- Download all latest device packages of products installed in the machine.
- Download newer versions of currently installed packages.
- Download the specified packages separated by commas.

You have to provide CCO credentials and the path of the server where the packages should be downloaded.

To schedule downloads:

---

**Step 1** Select **Common Services > Software Center > Schedule Device Downloads**.

The Schedule Downloads screen appears. Specify the Cisco.com user credentials, destination location, download policy, schedule, and job info.

**Step 2** Click **Accept** to put your settings into effect. To exit without making changes, click **Cancel**.

---

## Viewing Activity Logs

Activity Log records the jobs in Scheduled Downloads and Device Updates. It displays the activities that are carried out using Software Center.

There are two tables in the Activity Log.

- [Scheduled Job Details](#)
- [Event Logs](#)

**CISCO CONFIDENTIAL****Scheduled Job Details**

The Scheduled Job Details Log table records and shows the scheduled downloads that occurred in the server. You can view the application products and Job Id (if its scheduled download).

To view Scheduled Job Details log:

---

**Step 1** Select **Common Services > Software Center > Activity Log**.

The Activity Log dialog box displays the Scheduled Download Log and Event Logs tables.

**Step 2** Click on **Scheduled Job Details** to view Scheduled Download Logs.

---

**Event Logs**

The Event Log table shows the list of installation, uninstallation and immediate download activities carried out. Here the log records the product Name, description of the activity, the date of operation, the type of events and the status of the activity.

To view Event log:

---

**Step 1** Select **Common Services > Software Center > Activity Log**.

The Activity Log dialog box displays the Scheduled Download Log and Event Log tables.

**Step 2** Click on **Event Log** to view event logs.

---



**CISCO CONFIDENTIAL**

## CHAPTER **33**

# Using Common Incremental Device Support

---

Common Incremental Device Support (CIDS) is a mechanism allowing to add support for new types of devices to an NMS application without fully installing a newer version of the product. The CIDS layer encapsulates all mechanisms needed for this support.

EMBU products like CiscoView, DFM, RME, and Campus provide some form of incremental device support capabilities. Because each product has taken a different approach, there is a duplication of effort.

Instead of making each application IDS-capable individually and having a device package carry multiple pieces for each application separately, an architecture is proposed which introduces a new runtime component common for all applications. This architecture is CIDS and the common component is called Synchronous Device Interface (SDI).

SDI's purpose is to provide access to device-focused data and functionality by always accessing the live device (without caching). Hence the name, Synchronous Device Interface.

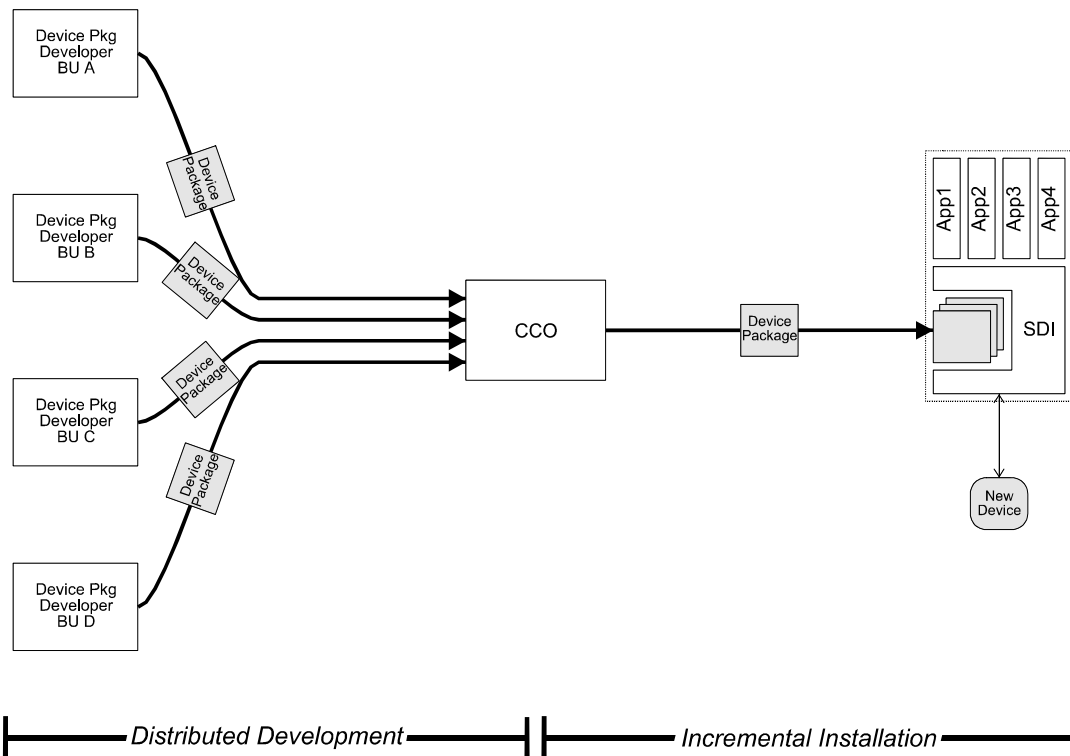
The new approach allows multiple applications to make use of the same device packages encapsulating device-specific functionality.

The new approach re-uses tools, processes, concepts, and resources developed for CiscoView. CiscoView incorporates a well-developed and tested model for incremental device support that has proven itself to be highly reliable and functional over a number of product releases.

CIDS allows you to:

- Add devices to an NMS application without fully installing a complete new version of the product.
- Re-architecture each individual application on top of a CIDS base layer.
- Have Synchronous Device Interface (SDI).
- Have de-centralized development.

[Figure 33-1](#) gives a high-level overview of CIDS device package development and installation.

**CISCO CONFIDENTIAL****Figure 33-1 High-Level CIDS Package**

For basic information on CIDS, see the “About the Common Incremental Device Support (CIDS) Component” section on page 29-5.

For more information about CIDS, see:

- The CIDS web site: <http://mspring-u10/cids/>
- The *Common Incremental Device Support (CIDS) for Appliances Functional Specification*, [ENG-100419](#)
- *Packaging CIDS*, [EDCS-184284](#)
- *SDI/ADI/UDI - Device Interfaces for Applications*, [ENG-113265](#)
- *SDI/ADI/UDI Design Specification*, [ENG-118149](#)
- *SDI/ADI/UDI - A Developer's Guide*, [EDCS-186733](#)
- The tools Java docs are available at:  
[http://wwin-nmbu/auto/cw/cdimages/uid1\\_0/daily/SOL\\_UID1\\_0\\_INTEGRATION\\_READY/kits/javadocs/](http://wwin-nmbu/auto/cw/cdimages/uid1_0/daily/SOL_UID1_0_INTEGRATION_READY/kits/javadocs/)
- The engine Java docs are available at:  
[http://wwin-nmbu/auto/em\\_storage/cw/cdimages/cids1\\_0/daily/SOL\\_CIDS1\\_0\\_LATEST\\_GOOD/kits/javadocs/](http://wwin-nmbu/auto/em_storage/cw/cdimages/cids1_0/daily/SOL_CIDS1_0_LATEST_GOOD/kits/javadocs/)

**CISCO CONFIDENTIAL**

## Understanding CIDS

From the customer's perspective there is just the Incremental Installation; Customer buys new hardware (entire new types of devices, or just new cards for existing devices). Customer wants to easily update his already running network management system to manage the new hardware.

From the NMS application development perspective there is the distributed development for device support: To avoid following every single new release from hardware Business Units within Cisco, the application development operates on device abstractions and delegates the actual implementation of these abstractions back to the hardware Business Units.

To understand a high level overview of CIDS architecture, see figure 1 of the CIDS system functional specification in EDCS (ENG-100419)

## SDI Component

SDI is the runtime component of CIDS. It provides the glue between the Network Management Application and the managed device. All the device specific information is then defined in a set of Abstraction Groups (AGs). The need is to develop a generic engine to drive the device management application for each device type based on the corresponding device description.

SDI:

1. Facilitates incremental device support by abstracting the device specifics from the application and the ability to drop in device packages.
2. Provides a common framework for all applications to provide incremental device support.
3. Provides support to the Application by performing device management operations and thereby eliminates the need for every application to duplicate this functionality.
4. Performs actual retrieval and modification of configuration information from/to the managed device.

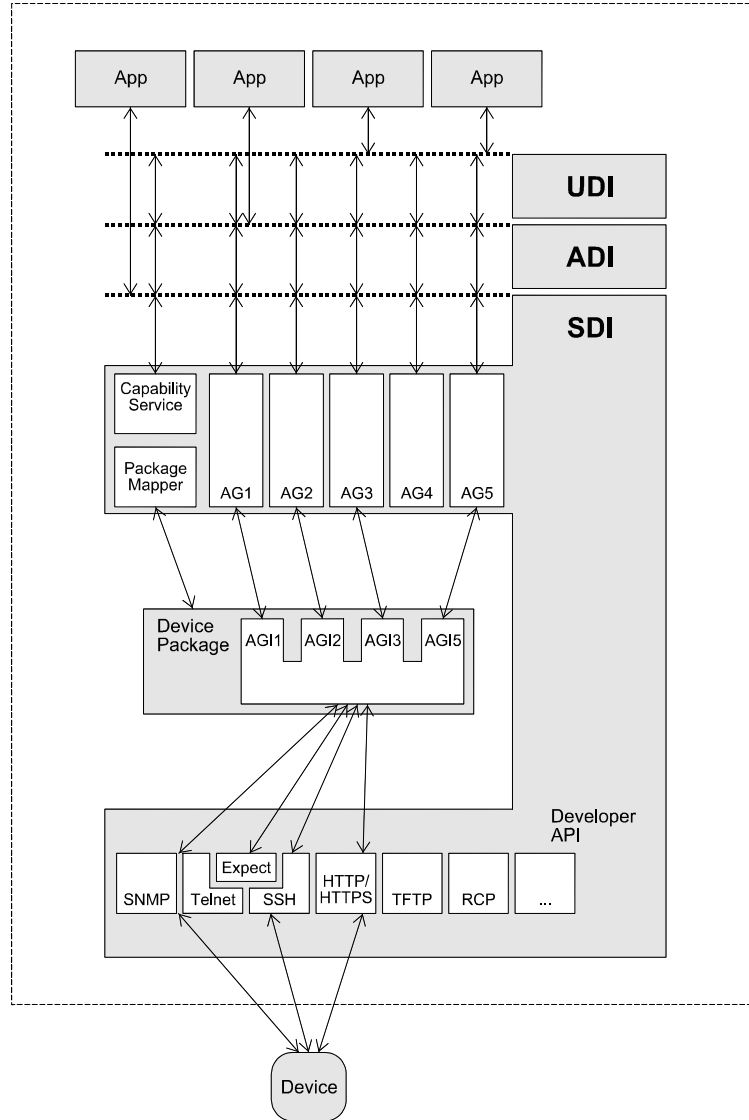
## Abstraction Groups

The Abstraction Groups (AGs) consist of a set of base classes. Each AG is represented by one Java package and consists of one or more Java classes. The AGs provide a generic interface for the Applications and shield the application developer from dealing with the device specific details. Application teams, working with the CIDS team, define Abstraction Groups in accordance with their needs.

**CISCO CONFIDENTIAL**

# Runtime Architecture

The figure below explains the runtime architecture of CIDS.



For more information on the CIDS architecture and the behavior, go to: <http://mspring-u10/cids/>





CISCO CONFIDENTIAL

## CHAPTER 34

# Using the Licensing APIs

---

The CWCS Licensing API allows you to:

- Install and update licenses.
- Retrieve license information.
- Access FLEXlm utilities that can be used to implement a wide variety of licensing models.

The following topics describe how to use the CWCS Licensing APIs with your application:

- [Understanding CWCS Licensing APIs](#)
- [Integrating CWCS Licensing APIs](#)

For basic information on the CWCS Licensing API, see the “[About CWCS Licensing](#)” section on [page 29-6](#).

For more information on CWCS Licensing APIs, see:

- [Licensing Requirements Document, EDCS-295207](#)
- [MDC CORE License Model, EDCS 153881](#)
- [CMF 2.3 System Function Specifications, EDCS-283137](#)
- [CMF Licensing Framework Functional Specification, EDCS-295256](#)

## Understanding CWCS Licensing

Software licenses are used to prevent unauthorized use of software products. Historical use of licensing technology in CWCS predecessor products included:

- **CMF-style licensing**, used in products such as RME and Campus Manager. This form of licensing uses proprietary technology and supports evaluation and purchased licenses, but did not address common requirements, such as feature- or size-based licensing.
- **FLEXlm-based licensing**, designed for VMS applications. This form of licensing was implemented in Core 1.0 (and supported in CMF up to version 2.2), but is inappropriate for general use, since it was designed specifically for VMS bundle use cases.

The current CWCS licensing:

- Continues support for the licensing components that existed in CMF 2.2. Applications that currently use that component can continue to do so without any changes.

**CISCO CONFIDENTIAL**

- Provides a facility for upgrade protection. This enables applications that currently employ CMF-style licensing to support upgrade licenses for customers who have a valid, purchased copy of the previous version of the application.
- Provides a framework for FLEXIm-based licensing. The framework supports feature- or size-based licenses through a simple API. Since some applications will have licensing requirements that are not addressed by this API, the framework provides FLEXIm tools that can be used to implement custom licensing schemes.

The CWCS licensing framework supports the following features:

- Licenses for evaluations, purchases, and non-revenue programs.
- Licenses that specify a resource limit, such as the number of devices that can be managed by an application.
- Licenses that grant the right to use an application or feature within an application.
- Temporary use of PIN to validate the use of a feature.
- Repository to store PIN and PAK (Product Authorization Key).
- API to install licenses and to retrieve license information, including PIN/PAK.
- License Administration GUI.
- FLEXIm toolkit.
- Backup for licenses.

License models other than those noted above such as node-locked licenses, floating licenses, counted licenses, etc., will not be directly supported in CMF 3.0. Applications that require such features must use FLEXIm toolkit to implement a licensing model that meets their requirements.

## Using Licensing UI

This topic provides information on the end-user interface of the License Information page:

To access the License Information page:

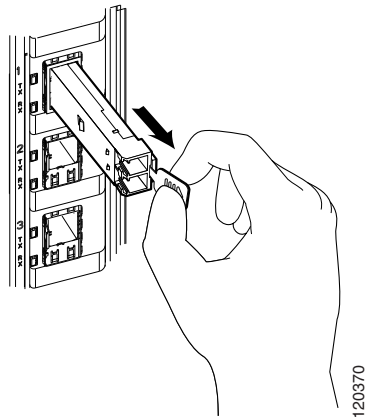
---

**Step 1** In the CiscoWorks Main Page, click **Common Services > Server > Admin**.

The Admin page appears.

**Step 2** Click **Licensing** link in the TOC.

The License Information page appears.

**CISCO CONFIDENTIAL**

The License Information page displays the details of the available licenses.

---

To update a license:

---

- Step 1** In the CiscoWorks Main Page, click **Common Services > Server > Admin**.  
The Admin page appears.
  - Step 2** Click **Licensing** link in the TOC.  
The License Information page appears with the details of available licenses.
  - Step 3** Click **Update**.  
The Select License File popup window appears. The CWCS installation directory is displayed in the License File field.
  - Step 4** Click **Browse** to select a license file.  
The Server Side File Selector popup window appears.
  - Step 5** Select a drive from the **Drive** list.
  - Step 6** Select the directory from the **Directory Content** list.
  - Step 7** Select the file from the list of files in the directory content area.  
The file must be a valid license file with read permissions for the user.
  - Step 8** Click **OK**.
-

**CISCO CONFIDENTIAL**

## Understanding CWCS Licensing APIs

The Licensing framework supports Java interface to install and retrieve license information. The licensing API supports the following operations:

- Retrieve names of all bundles for which a license key/PIN has been installed.
- Install PIN/PAK combination.
- Authorize PIN/PAK associated with an application or a bundled solution.
- Retrieve PIN/PAK associated with an application or a bundled solution.
- Retrieve all installed PIN/PAK.
- Install license keys.
- Authorize license key(s) for an application or a bundled solution.
- Retrieve license data for an application.

License data would contain information such as the installation date, expiration date, license type, number of devices that can be managed, and PAK/PIN or license key(s) associated with the application.

## CWCS Licensing Classes

The CWCS Licensing classes are:

- [LicensedFeature](#)
- [LicenseManager](#)
- [LicensePAK](#)

### LicensedFeature

LicensedFeature represents a feature or application that has license (such as RME), in addition to a name, version, expiry date and the number of licenses.

Instances of this class contains information on whether the feature has an evaluation, upgrade, purchased, or a special kind of license. Since feature licenses may be delivered via a FLEXlm license file or a PIN, instances of this class may also be queried for the source of licensing information.

The following table provides information on the methods used by LicensedFeature:

**Table 34-1** *Methods Used by Licensed Feature*

Method	Description
installTime	Retrieves the time at which the license was installed.
bundleName	Retrieves the name of the bundle that the PAK/PIN licenses.

**CISCO CONFIDENTIAL****LicenseManager**

LicenseManager helps applications to manage licenses. Applications based on CMF that need to enforce license restrictions must use this class to install and retrieve licenses.

The following table provides information on the methods used by LicenseManager.

**Table 34-2** *Methods Used by LicenseManager*

<b>Method</b>	<b>Description</b>
addLicense	Adds license keys in a FLEXlm license file.
getFeature	Gets information on all versions of a licensed feature or the specified version of a licensed feature.  Returns null if no license for the feature has been installed.
getAllFeatures	Gets information on all licensed features.  Returns null if no license is installed.
addPAK	Adds a PAK/PIN combination.
getPAK	Retrieves all installed PIN/PAK combinations or the registered PIN/PAK for a bundle.  Returns null if no PIN/PAKs are installed.

**LicensePAK**

LicensePAK represents a PAK/PIN combination and the time at which the PAK/PIN was installed.

The arguments for this class are PAK and PIN associated with the license. The installation time of the LicensePAK is set to current time.

The following table provides information on the methods used by LicensePAK.

**Table 34-3** *Methods used by LicensePAK*

<b>Method</b>	<b>Description</b>
getPIN	Retrieves the PIN associated with the license.
getPAK	Retrieves the PAK associated with the license.
installTime	Retrieves the time at which the license was installed.
bundleName	Retrieves the name of the bundle that the PAK/PIN licenses.

**Error Codes Generated by APIs**

The following table describes the error codes generated by the API:

**CISCO CONFIDENTIAL****Table 34-4 Licensing Errors**

Error Code	Description
LicenseError.CorruptLicenseFile	Exception to indicate a corrupt license file.
LicenseError.EvalExtension	Exception to indicate an extension of the evaluation period.
LicenseError.ExpiredLicense	Exception to indicate the expiry of a license.
LicenseError.MalformedPIN	Exception to indicate an incorrect PIN.

## Integrating CWCS Licensing APIs

This section provides information on integrating the licensing APIs with your application.

### JavaDoc

The Javadoc for the License API can be found at following URL.

<http://nmtgre.cisco.com/auto/embueng/License/>

### License Installation

Licenses can be installed during the installation of a bundle, or at a later time.

The license installation UI does not appear during the Common Services 3.0 installation. The UI appears only for the first application in the bundle that is installed.

The application install code must verify the existence of a valid license for that application. If a valid license is missing, the application prompts to enter the license at the installation time.

If the first application installation contains a license key that has entries for all the applications in the bundle, subsequent application installations do not have to prompt for licenses.

- The license file being installed is processed in addition to the existing license files.
- If no valid license entry is provided for an application, the application will run in EVAL mode.
- If a PIN is provided for the application, the application will run in NAG mode until a valid license is provided. For more information on NAG mode, see [CMF Licensing Framework Functional Specification, EDCS-295256](#)
- If an upgrade license is provided, You must provide a Proof-of-Purchase for that application by running the CMF-provided script with appropriate arguments (validate\_upgrade.exe).

Version of the FLEXlm libraries : Version 9.2

## CISCO CONFIDENTIAL

### PAK and PIN

PAK and PIN are processed in pairs—while the PAK carries no information that is useful to CMF, a PIN is associated with a product or an application, and encodes the type of license (evaluation, upgrade, or purchased) the user has for that product.

1. License Key/PIN is invalid—In this case, since there is no meaningful association between the license key or the PAK/PIN combination and a product, the input is rejected and an error message is displayed with the cause for the failure.
2. License Key/PIN indicates an evaluation license—Scenarios that could be potentially destructive if the current installation is run over of an existing installation of the previous version of the product:
  - The user has only an evaluation license of the previous version—The evaluation license represented by the PIN is invalid and the user will find the product unusable after completing installation.
  - The user has a valid purchased license of the previous version of the product—The user can use the latest installed version in evaluation mode. However, the product would become unusable at the end of the evaluation period, unless a purchased license is installed later.
3. License Key/PIN indicates an upgrade from the previous version—When the user has only an evaluation version of the previous version, installing an upgrade license is equivalent to installing an evaluation license, which is not allowed.
4. License Key/PIN indicates a purchased license—Previously installed versions are irrelevant since the user has a valid, purchased license for the current version.

Since cases 2 and 3 can be potentially destructive, the user is warned, that continuing with the installation may render the product unusable, and the user should be given the option to cancel the installation. If the user proceeds with the installation, the License Framework will process all the license keys, PAKs and PINs that were entered by the user. In the case of fully purchased licenses/PINs, the license will be installed and no further processing is required during product installation. In all other cases, the licenses will be maintained in a state where their use is permitted only after authorization by a product or application.

**Note**

---

It is the responsibility of the product installation to determine whether a license key/PIN presented by the user is to be treated as a valid license and invoke an API that will authorize the license.

---

### Handling Multiple Licenses

Applications may be licensed as part of an application bundle or separately. It is important to determine the effect of installing a license for an application for which a license has already been installed. In the following section, a PIN that is not superseded by the license key it represents is treated the same as the license key. We consider the following cases here:

**CISCO CONFIDENTIAL**

Scenario	Description
Installing an evaluation license over a previously installed license.	This operation is invalid and the new evaluation license will be ignored.
Installing an upgrade license over an evaluation license.	The upgrade license will supersede the evaluation license.  If a fully purchased license for a previous version of the application is installed, the installed license will be upgraded to the newer version of the application. Otherwise, the upgrade license will be considered <i>incomplete</i> and the application that uses this license will function in NAG mode.  For more information on Nag mode, see <a href="#">License Requirements Document, EDCS-295207</a> .
Installing an upgrade license over a fully purchased license	If the upgrade license is for the same or an earlier version of the application, as compared to the existing one, the upgrade license will be ignored. If the upgrade license is for a more recent version of the application, the installed license will be upgraded to the newer version of the application.
Installing a purchased license over an evaluation license.	Purchased license is installed.
Installing a purchased license over a previously installed purchased license.	Increases the device count.  Since purchased licenses do not have an expiration date, the only effect of this operation is to increase the device count, if any.  For example, if an application had license for 1000 devices, and a new license with a device count of 500 is installed for the application, the number of licenses for that application increases to 1500 devices.

## Interpreting PIN

PIN allows a customer to continue using the product and provide information to the licensing component before the actual license key is entered. PIN provides the following information to the licensing component:

- Bundle name
- Applications and their versions in the bundle
- Allowable device limits per application
- Type of installation (Such as New, Upgrade, SEVT, Network Academy)

PIN does not contain more information than the license key. The license key contains all information that the PIN contains, and more.

The PIN alone does not provide any useful information regarding application name, version, and device limit and installation type. Each application code in the PIN should have a mapping file to get it mapped with an application. To enable this functionality, the evaluation license of each application is used. The



## CISCO CONFIDENTIAL

evaluation license of each image has the application name, version and associated application code. The application code in license file is matched with the application code in the PIN to get the details about the application.

The PIN is not a substitute for the license key. The main function of the PIN is to allow the customer to continue using the product in the absence of a license key. This is because very often the user installing the product is not the one who purchased the product and obtains the license key. However, the customer has to enter the license key to move the product from the nagging mode to the full function mode.

If the customer does not enter the actual purchased license key within 90 days, and if a PIN is entered at install time, the product will always function in a NAG mode. In NAG mode, after 90 days, if the PAK and PIN are present, NAG messages will start appearing. If the PAK and PIN are not entered, i.e. if the product is in EVAL mode, the product stops functioning after 90 days.

The customer can enter the license key at install time or using the desktop GUI. It is recommended that the customer enters the actual license key rather than the PIN at install time. The PIN is only a fallback mechanism.

## PIN Format

The format of the PIN has been designed to encode all the information required by Common Services licensing. A PIN can encode license information for a bundle or an arbitrary collection of applications.

PIN specification:

- A PIN consists of 32 characters with hyphens separating 8 character sequences. Valid characters in a PIN are [2-9][A-Z] with the exception of the characters **I** and **O**.
- The first eight characters describe the characteristics of a bundle. Bundles are viewed as collections of applications. Hence, a PIN representing an arbitrary collection of applications is treated in the same manner as a bundle.

Description of a bundle can be further broken down as follows:

- Characters in position 1-4 name the bundle. If the name of a bundle is only 3 characters long, the unused character position is filled with 9.
- Characters 5-6 encode the bundle version.
- Character 7 encodes the license type (Permanent, Upgrade, etc).
- Character 8 encodes whether the resource counts in the PIN are cumulative or absolute.
- The last character in the PIN encodes the number of applications in the bundle. Since 0 and 1 are not valid characters, the value for this character is 2 + the number of applications.
- Characters starting from position 8, encodes applications in the bundle, with three characters used for each application. Of the three characters used for an application, the first two encode the name of the application and its version. DNMBU marketing will maintain a table of application codes and their mapping to licensed applications. For example, RME 4.0 may be represented by the code 22, DFM 2.0 by the code 23, etc. The third character used for an application encodes the resource count. All characters that are valid in PINs are mapped to preset resource counts; the character that is used to denote the resource count for an application is that whose associated count is closest to (but not less than) the application resource count.
- The character at position 31 encodes a checksum for all the characters described above.
- All unused character positions are filled with valid characters selected at random.

## CISCO CONFIDENTIAL

- Since 10 character positions are used to encode bundle information, the number of applications and the checksum, only 22 characters are available to encode the applications in a bundle. This implies that PINs can be used to encode bundles only when they have 7 or less number of licensed applications.

## Understanding License Framework

The Licensing framework provides a way for licensing the evaluation version. The FCS image can be used as both evaluation version and purchased version.

Applications that need to use the Licensing framework must get the application mapping code from CWCS marketing team. (The application code is unique for each application and version. For example, RME 4.0 has an application code 22). Each application must get a static evaluation license file from CMF team. (The static license file contains application code map for the application, and device limit for the Evaluation mode).

This static evaluation license file should be part of the applications image. This must be under `<ApplicationImage>/disk1/eval/<AppName.lic>`. This license file is used during install, if the user selects Eval mode or PIN/PAK mode. The purchased license is generated by SWIFT based on the serial number or PAK.

When an application is installed, the framework will check for available license in the repository based on the Application name, version, and Application code.

If a matching entry is found either in license file or PIN, the framework does not prompt for license details. Otherwise, the user is prompted for license details.

The license file or PIN is validated and the installation proceeds.

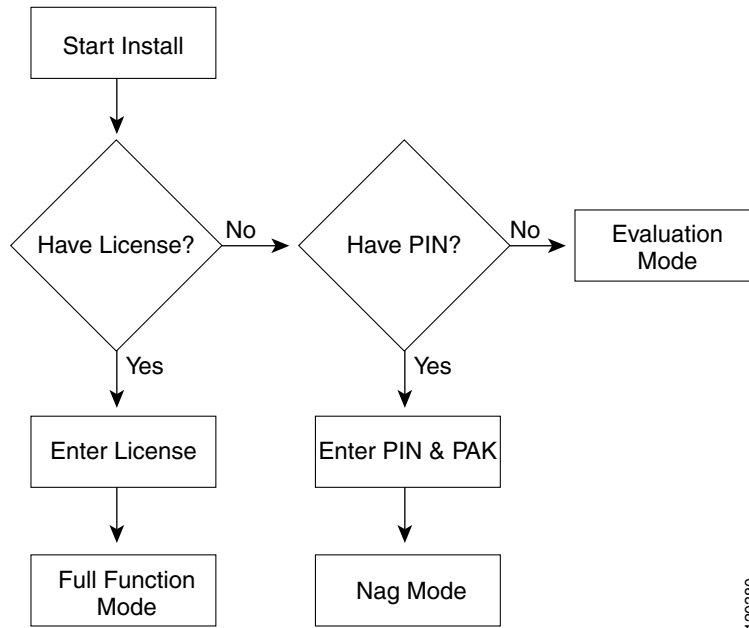
The license repository is `NMSROOT/etc/licenses`. When a new license is added, the information is added to the repository with `.lic` extension.

The PIN/PAK details are stored in `pinpak.data` file.

## Flowcharts

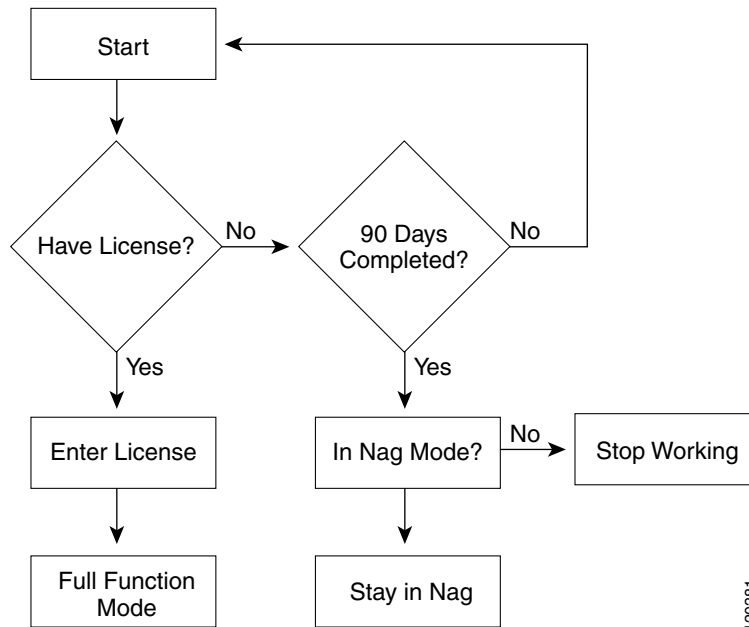
License Key and PIN entry during installation:

**CISCO CONFIDENTIAL**



120380

License Key and PIN entry within a 90 day period after installation:



120381

CISCO CONFIDENTIAL

# Using Licensing Framework With Applications

To use the Licensing Framework provided by CWCS, the application must modify the following sections:

- [Install](#)
- [CW Home Page](#)
- [Runtime Calls](#)

## Install

License install framework is part of ITOOLS.

To use the licensing install framework:

**Step 1** Create a PRODUCT\_INFO TAG in the respective disk.toc with application name and version.

Example: PRODUCT\_INFO=rme 4.0



**Note** Application name must be in lower case.

**Step 2** Get application mapping code for the application from CWCS marketing.

**Step 3** Create an evaluation license for the application and place it under <ApplicationImage>/disk1/eval/.

The license file name should read: <application name>.lic

## CW Home Page

Applications must verify the license type. If the license type indicates that the application is for non-revenue programs, the application should display the message “The product is not licensed for commercial use.”

Examples of non-revenue programs: SEVT, NFR, NA

API Call for getting the license type for an application:

API Call	Description
LicensedFeature.licenseType( )	The call returns the type of license for that application.

## Runtime Calls

To query licensing details at runtime:

**Step 1** Import the following:

```
import com.cisco.nm.license.client.*;
import com.cisco.nm.license.util.*;
```

**CISCO CONFIDENTIAL**

- Step 2** Create instances of:
- ```
LicenseManager lm = new LicenseManager();
LicensedFeature licFeature = lm.getAllFeatures("Appname", "Version");
```
- Step 3** Use APIs to query the license details at runtime.



Note Licensing framework use Log4J for logging. Application using License class should handle logging properly.

License SDK

The License framework will include an SDK containing FLEXlm tools and documentation, so that the applications can implement licensing models other than the one supported by CMF. The SDK contains:

- Programs to implement a license server (lmgrd and vendor daemon)
- Runtime libraries and relevant C language header files
- Tools for administering licenses and generating evaluation licenses

Data Architecture

Licenses, PINs and PAKs that are installed using the License GUI or API can be retrieved using the License API. No other specifications are made regarding the internal representation of the license data or the mechanisms used to persist such data.

License File Format

The license file format specified in [MDC Core License Model Document, EDCS-153881](#) is used for licensing other bundle applications.

In addition to the file format supported in Common Services 2.2 (as described in EDCS-153881), a new license file format will be used to support the features in Common Services 3.0. Applications that intend to use the features described in this document are required to use the new license file format described in section License File Format for Common Services 3.0.

The license file has a fixed structure. Words in **CAPITAL BOLD** are key words, unique to FLEXlm. Words in *Italics* should be replaced by the actual data. Other wording should be there in order for FLEXlm to be able to parse the syntax.

```
VENDOR cisco
INCREMENT licenseInfo cisco 1.0 expirationDate uncounted \
VENDOR_STRING=DeviceLimit HOSTID=Any \
NOTICE=DemoLength SIGN=0
INCREMENT licenseUser cisco 1.0 expirationDate uncounted \
NOTICE=PAK HOSTID=Any SIGN=0
INCREMENT PIX cisco 1.0 expirationDate uncounted \
NOTICE=PAK VENDOR_STRING=DeviceLimit HOSTID=Any SIGN=0
INCREMENT IOS cisco 1.0 expirationDate uncounted \
NOTICE=PAK VENDOR_STRING=DeviceLimit HOSTID=Any SIGN=0
```

CISCO CONFIDENTIAL

```

INCREMENT IDS cisco 1.0 expirationDate uncounted \
NOTICE=PAK  VENDOR_STRING=DeviceLimit  HOSTID=Any  SIGN=0
INCREMENT C3K cisco 1.0 expirationDate uncounted \
NOTICE=PAK  VENDOR_STRING=DeviceLimit  HOSTID=Any  SIGN=0

```

The first INCREMENT is followed by the keyword *licenseInfo* to identify that this feature line will describe the general license information.

| Field | Description |
|-----------------------|--|
| <i>cisco</i> | Name of the Vendor. |
| <i>1.0</i> | Version. FLEXlm supports supplying versions between 1.0 and 2.0 inclusive. As other versions of CORE come out, we can increment the 1.0 version. |
| <i>expirationDate</i> | This can either be the keyword “permanent” to indicate that the license is purchased and never expires, or can be a date to indicate that this is a demo license key. The expiration date is the date on which the license disk becomes invalid and further installations with this license key are invalid. The date format is dd-mmm-yyyy, where dd and yyyy are numeric, and mmm is the three letter abbreviation for the month. For example, Jan 5, 2002 would be represented as 05-jan-2002 and not 5-jan-2002. |
| <i>uncounted</i> | Required as the customer will not be running a FLEXlm license server. |
| <i>DeviceLimit</i> | The number of devices licensed for an application. If unlimited, use -1. |
| <i>DemoLength</i> | The number of days a demo license disk is valid for after installation. In case of a purchased license this value is -1 to indicate that the license does not expire. |
| SIGN=0 | After running lmcrypt, this will hold the signature associated with the INCREMENT line.

INCREMENT line with <i>licenseUser</i> is used to store information on the user to whom the license was issued. |
| <i>PAK</i> | Product Authorization Key. |
| | Additional INCREMENT lines are for devices. Each additional device type has an INCREMENT line corresponding to it. |
| <i>ExpirationDate</i> | Usually the same as the expiration date for the whole license disk. |
| <i>DeviceLimit</i> | If unlimited, -1 is used. |
| # | Comment lines are started with this symbol. |

CISCO CONFIDENTIAL**License File Format for Common Services 3.0**

Common Services 3.0 uses a new license file format.

Sample license file for an evaluation license for CDOne 3.0:

```
INCREMENT cdone cisco 3.0 31-dec-2004 uncounted \

VENDOR_STRING=<LicType>Evaluation</LicType><Code>ZZ</Code><Count>10</Count><CountType>Absolute</CountType><XINFO>LMS30</XINFO> \
  HOSTID=ANY \
  NOTICE="<LicFileID>lms</LicFileID><LicLineID>0</LicLineID> \
  <PAK>dummyPak</PAK>" SIGN=48CCDF82DDF6
```

Each application in the bundle is represented by an INCREMENT line that specifies the name and version of the application. The first line in the file describes the bundle itself and does not contain any information that is useful to the applications. In addition to the feature name and version, Common Services licensing uses the fields expiry date (set to 31-dec-2003 in this example) and VENDOR_STRING to store the information it requires.

| Field | Description |
|--------------------------------------|---|
| expiry date | The specified expiry date for the evaluation licenses is not currently used but may be used in future to determine a date beyond which the evaluation license may not be installed. |
| Date | For Evaluation License File. |
| <i>permanent</i> | For all other files. |
| VENDOR_STRING | This field can be used by a vendor to customize the license file. |
| License type(using the tag LicType) | The value for this tag can be one of Evaluation, SEVT, NFR or NA.

Permanent licenses and upgrade licenses need not encode this information. |
| Resource count (using the tag Count) | The number of devices (or any other resource) that the application is licensed for.

This value is -1 when no constraints apply. |

CISCO CONFIDENTIAL

| Field | Description |
|--|--|
| Interpretation of resource count (using the tag CountType) | <p>Indicates how the value specified for resource count is to be interpreted.</p> <p>Values for this tag are:</p> <ul style="list-style-type: none"> • Cumulative— the specified count is to be treated as an addition to any prior license for combination of application and version. • Absolute— the specified count should override any prior license. |
| Application code (using the tag Code) | <p>The code assigned to this combination of application and version by DNMBU marketing.</p> <p>The licensing component requires this tag for deciphering PINs. This tag is present only in evaluation license.</p> |

Alternate License File Format

License files issued by Swift will employ an alternate license file format to represent license information. This format uses other FLEXlm license file fields as follows to represent the information stored in the VENDOR_STRING field in the format described in the previous section.

Sample license file for the alternate format:

```
SERVER
VENDOR cisco
UPGRADE CDONE cisco 2.2 3.0 permanent 10 \
  VENDOR_STRING=<XINFO>LMS30</XINFO> \
  NOTICE="<LicFileID>lms</LicFileID><LicLineID>0</LicLineID> \
  <PAK>dummyPak</PAK>" SIGN=535525A647B6
FEATURE RME cisco 4.0 permanent 1500 \
  VENDOR_STRING=<XINFO>LMS30</XINFO> \
  NOTICE="<LicFileID>lms</LicFileID><LicLineID>0</LicLineID> \
  <PAK>dummyPak</PAK>" SIGN=7078C85E40E8
FEATURE DFM cisco 2.0 permanent 1000 \
  VENDOR_STRING=<XINFO>LMS30</XINFO> \
  NOTICE="<LicFileID>lms</LicFileID><LicLineID>2</LicLineID> \
  <PAK>dummyPak</PAK>" SIGN=52A22CF20DF4
FEATURE CampusManager cisco 4.0 permanent 500 \
  VENDOR_STRING=<XINFO>LMS30</XINFO> \
  NOTICE="<LicFileID>lms</LicFileID><LicLineID>3</LicLineID> \
  <PAK>dummyPak</PAK>" SIGN=333AD18EF768
```


CISCO CONFIDENTIAL

| Field | Description |
|------------------|--|
| License type | License files issued by Swift represent either Permanent or Upgrade licenses. UPGRADE lines represent upgrade licenses while FEATURE and INCREMENT lines represent permanent licenses.

An upgrade license results in the device/resource counts from previous versions to be carried over. When an upgrade license is used to increase the resource count, an additional INCREMENT or FEATURE line will be present to indicate the increased count. |
| Resource | The count will be encoded in the license count field of FLEXlm license files. The value of this field is uncounted when no restrictions apply. |
| Count type | Resource counts in INCREMENT lines are Cumulative while the counts in FEATURE lines are Absolute. |
| Application code | Permanent and Upgrade licenses will not contain an application code. |

Proof-of-Purchase (POP)

Proof-of-Purchase (POP) will be obtained when you enter an upgrade license key. The customer will not be able to install a product if they have paid only for an upgrade license, unless the customer has got a the license for a previous version.

One way of ensuring that a customer has purchased a previous version of the product involves checking for the previous version installation CD. The customer will only be required to use a CD as a last resort, even in a case where the upgrade is done on a new machine and the previous version of the product is one another machine. If a CD check is ultimately needed, the licensing component must *not* require the user to remove a CD that is being used to install the upgrade, replace it with a CD for the previous version to do the POP, then reinsert the product upgrade CD.

The licensing component will perform a POP check by checking the license file of the previously installed version of a definitive product for which the upgrade license has been purchased.

Versions of applications that are not based on CS 3.0 such as RME 3.4, RME 3.5, DFM 1.1, DFM 1.2, Campus Manager 3.2, and Campus Manager 3.3 have license files to help implement the CMF-style licensing. These files are called rme.xml, dfm.xml and cm.xml respectively for RME, DFM and Campus Manager respectively. These files contain an EXPIRY line with two possible values: NEVER (for purchased license) and 90 (for an evaluation license).

One application per bundle will be used to identify which bundle the customer had purchased. The following table shows what needs to be checked per bundle:

CISCO CONFIDENTIAL**Table 34-5 Files to be Verified by Applications**

| Bundle/Licensing Component Consumer | Definitive Application | Files | Comments |
|-------------------------------------|--------------------------|---------------------|--------------------------|
| LMS | RME | rme.xml | |
| RWAN | ACLM | N/A | Expected to be EOS'd. |
| ITEM | ITM | itm.xml | |
| CVM | CVM | cvm.xml | |
| EMS | RME | rme.xml | |
| Cable Mgr | RME | rme.xml | |
| QPM | QPM | qpm.xml | |
| SNMS | RME | rme.xml | |
| VMS | Core | N/A | Use license in database. |
| ACLM/IPM add-on to LMS | ACLM or IPM respectively | aclm.xml or ipm.xml | |

In-Place Upgrade

Install can automatically check for Proof-of-Purchase if the customer enters an upgrade license key or PIN, the proof of whether the customer has a purchased previous version is available immediately from the installed version.

Remote Upgrade

Install must prompt for Proof-of-Purchase. If the customer enters an upgrade license key or PIN, the proof of whether the customer has a purchased previous version must be verified.

- **Upgrade License Key is entered at install time**

At the end of installation, a message similar to the one below is displayed:

```
You have entered an upgrade license key. Please run the program
<NMSROOT>/bin/validate_upgrade.exe to validate that this is an upgrade.
```

Until the user runs the above program, the product will run in NAG mode.

- **Upgrade PIN entered at install time**

At the end of install, a message similar to the one below will be displayed:

```
"You have entered an PIN and not a license key. The product will continue to work in
nag mode from the date of installation. However, please obtain a valid license key
from CCO in order to make the product fully functional.
```

```
After obtaining the license key, load it into CiscoWorks by clicking on: Common
Services > Server > Admin > Licensing.
```

```
After entering the license key, please run the program <NMSROOT>/bin/
validate_upgrade.exe to validate the upgrade."
```

Until the user runs the above program, the product will run in NAG mode.

- **Upgrade License Key entered through GUI After the install**

CISCO CONFIDENTIAL

After entering the license through the GUI, a message will be printed similar to the following:

```
You have entered an upgrade license key. Please run the program
<NMSROOT>/bin/validate_upgrade.exe to validate the upgrade.
```

Until the user runs the above program, the product will run in NAG mode.

License CLI**Note**

This section is applicable only to previous VMS based license.

The following programs are provided to support the installation of licenses:

Table 34-6 License Program CLIs

| License CLI | Description |
|-----------------|--|
| validatelicense | Verifies whether the supplied license is valid and whether it can be supported by CMF Licensing
Location:
NMSroot/MDC/bin |
| addlicense | Upgrade customers can use this utility if the upgrade check fails during the install.
Location:
NMSroot/bin |

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

GLOSSARY

A

- ACLM** Access Control List Manager. ACL Manager dramatically reduces the time needed to develop new filters and maintain existing traffic filters in large-scale deployments of Cisco devices.
- aggregate table** See discovery.
- ANI** Asynchronous Network Interface (ANI) is a mediation layer between the network devices and client applications that provides for the discovery, inventory, and topological computations of networks and their devices.
- AniAggregateTable** A collection of SMFCContainers—a global data structure that stores the contents of SMFCContainers and their names. See also discovery.
- ANI Server** Asynchronous Network Interface Server. A Java application that performs multiple tasks, including performing network discovery using Cisco Discovery Protocol (CDP) and Integrated Local Management Interface (ILMI), processing SNMP requests, and serving as a middle-tier server for the client applications that need to query and set the state of the network or network devices
- API** Application Programming Interface. A language and message format used by an application to communicate with the operating system and other services (such as a database management system or communications protocol).
- ASA60** Adaptive Server Anywhere 6.0. The new name for SqlAnywhere.
- ATM** Asynchronous Transfer Mode. International standard for cell relay in which multiple service types (such as voice, video, or data) are conveyed in fixed-length (53-byte) cells. Fixed-length cells allow cell processing to occur in hardware, thereby reducing transit delays. ATM is designed to take advantage of high-speed transmission media such as E3, SONET, and T3.
- ATM fabric** A set of ATM switches interconnected by ATM links such that any switch in the fabric can be reached from any other switch in the fabric by traversing one or more ATM links and optionally one or more ATM switches in the fabric. The fabric contains both ATM switches and all ATM links that are connected to those ATM switches, including links to edge devices, but not the edge devices themselves.

C

- CCO** Cisco Connection Online Web site. Used to access customer service and support.
- CDP** Cisco Discovery Protocol. Media- and protocol-independent device-discovery protocol that runs on all Cisco-manufactured equipment: routers, access servers, bridges, and switches. Using CDP, a device can advertise its existence to other devices and receive information about other devices on the same LAN or on the remote side of a WAN. Runs on all media that support SNAP, including LANs, Frame Relay, and ATM media.

CISCO CONFIDENTIAL

| | |
|------------------------------|---|
| client | Node or software program (front-end device) that requests services from a server. |
| client/server | Term that describes distributed computing (processing) network systems in which transaction responsibilities are divided into two parts: client (front end) and server (back end). Both terms (client and server) can be applied to software programs or actual computing devices. |
| CMF | Common Management Foundation. Predecessor system of CWCS . |
| community strings | Text strings that act as a password to authenticate messages sent between the network management station and devices containing an SNMP agent. Community strings allow you to limit access to network devices. |
| CORBA | The Common Object Request Broker Architecture (CORBA) is an industry standard middleware architecture developed and maintained by the Object Management Group (http://www.omg.org). CORBA services act as communication mechanisms for developing distributed applications. CORBA is platform and language neutral, which means that a C application running on a PC can communicate with a Java application running on Solaris. |
| CSV | Comma Separated Values. An interchange file format typically used for exporting and importing spreadsheets or other tables. Each line in the ASCII file represents a row of data from a table. Each line contains the data elements from a row of the table, with individual table values separated by comma characters. |
| CWCS | A collection of subsystems, execution environments, engines, and shared code libraries, representing a software platform that provides services to web-based network management applications. The end user documentation refers to CWCS as the CiscoWorks Server. |
| CWCS Base Services | First tier, entry-level CWCS components necessary to support a web-based application. These components include the web server, CWCS security, the servlet engine, and JRE. |
| CWCS Network Services | Third-tier CWCS components that add discovery and other network services. |
| CWCS System Services | Second-tier CWCS components that add services such as EDS, JRM, and the database engine. |

D

| | |
|-----------------------|--|
| daemon | A process that runs unattended to perform a standard service. |
| Daemon Manager | A CWCS component that initiates, monitors, and controls application processes. Also known as Process Manager . |
| daemon process | A process that is started by the root user or the root shell and can be stopped only by the root user. Daemon processes generally provide services that must be available at all times, such as sending data to a printer. |
| DBD | Database Driver. A Perl interface module that implements DBI functions for a vendor-specific database. |

CISCO CONFIDENTIAL

| | |
|------------------------|---|
| DBI | Database Interface. A public domain Perl module on UNIX platforms that provides a vendor-independent API for accessing relational databases. A subset of DBI is implemented by Cisco on NT platforms using WIN32::ODBC for compatibility with UNIX. |
| device adaptor | Subclass that allows network management applications to access new devices and new functionality in existing devices without a complete code rewrite. The device adaptor model is based on an inheritance tree that extends base class functionalities. |
| device class | A group of SNMP-based device types that support the same MIB information. |
| device conflict | The relationship between an unmanaged device and a managed device when their definitions share the same device access information (either the same DNS domain name and network hostname combination or the same IP address) but differ in one or more of the access information elements. |
| discovery | Process performed by the ANI Server to locate and identify the devices and topology of the network. |
| DLL | Dynamically Linked Library. A library linked to application programs when they are loaded or run rather than as the final phase of compilation. DLLs allow several tasks to share the same block of library code. |
| DNS | Domain Name System. System used in the Internet for translating names of network nodes into addresses. |
| DSN | Data Source Name. A definition file or NT registry key used by the ODBC to obtain database connection parameters such as port numbers, engine and database name, database file, start line parameters, and so on. |
| DTD | Document Type Definition. Contains a formal definition of a particular document type. Although it is not mandatory, specifying a DTD allows parsers to validate an XML file. |

E

| | |
|-------------------|---|
| EDS | Event Distribution System. Event management software that provides the means for sending messages from one process to another in a networked and distributed environment. |
| ELAN | Emulated LAN. ATM network in which an Ethernet or Token Ring LAN is emulated using a client-server model. ELANs are composed of an LEC, an LES, a BUS, and an LECS. Multiple ELANs can exist simultaneously on a single ATM network. ELANs are defined by the LANE specification. |
| Essentials | Resource Manager Essentials (Essentials), one of the major components of CiscoWorks, enables the deployment, monitoring, and configuration of devices across a network. Resource Manager Essentials includes the following applications: Inventory Manager, Change Audit, Device Configuration Manager, Software Image Manager, Availability Manager, Syslog Analyzer, and Cisco Management Connection. |
| EvalGroup | A pool of threads that ANI uses for the parallel evaluation of tasks. |

F

CISCO CONFIDENTIAL

| | |
|------------------|--|
| fabric | See discovery. |
| framework | A set of tools that provide for installation and uninstallation of Cisco product and allow you to build installable CD images. |

G

| | |
|------------|---|
| GUI | Graphical User Interface. User environment that uses pictorial as well as textual representations of the input and output of applications and the hierarchical or other data structure in which information is stored. Conventions such as buttons, icons, and windows are typical, and many actions are performed using a pointing device (such as a mouse). |
|------------|---|

H

| | |
|-------------|---|
| HTTP | Hypertext Transfer Protocol. The protocol used by Web browsers and Web servers to transfer files, such as text and graphic files. |
|-------------|---|

I

| | |
|----------------------------|---|
| IDL | Interface Definition Language. An Open Software Foundation standard used to define discovery interfaces. |
| ILMI | Integrated Local Management Interface. Standard discovery protocol used on ATM networks. |
| installable unit | A package or group of packages which are installed or uninstalled at once. |
| IP address | 32-bit address assigned to hosts using TCP/IP. An IP address consists of a network number, an optional subnetwork number, and a host number. The network and subnetwork numbers together are used for routing, while the host number is used to address an individual host within the network or subnetwork. Also called an Internet address. |
| Integration Utility | See NMIM . |

J

| | |
|-----------------|--|
| Jconnect | Sybase implementation of JDBC |
| JDBC | Java Database Connectivity. Java API for accessing relational databases. |
| JNI | Java Native Interface. A native Java API that allows Java code running inside a Java Virtual Machine to interact with applications written in other programming languages such as C and C++. |
| JRE | Java Runtime Environment. |
| JRM | Job and Resource Manager. A CWCS component that allows applications to schedule jobs and lock resources. |

CISCO CONFIDENTIAL

L

- LANE** LAN emulation. Technology that allows an ATM network to function as a LAN backbone. The ATM network must provide multicast and broadcast support, address mapping (MAC-to-ATM), SVC management, and a usable packet format. LANE also defines Ethernet and Token Ring ELANs.
- LECS** LAN Emulation Configuration Server. Entity that assigns individual LANE clients to particular ELANs by directing them to the LES that corresponds to the ELAN. There is logically one LECS per administrative domain, and this serves all ELANs within that domain. See also [ELAN](#).
- LES** LAN Emulation Server. Entity that implements the control function for a particular ELAN. There is only one logical LES per ELAN, and it is identified by a unique ATM address. See also [ELAN](#).

M

- MIB** Management Information Base. Database of network management information that is used and maintained by a network management protocol such as SNMP or CMIP. The value of a MIB object can be changed or retrieved with SNMP or CMIP commands. MIB objects are organized in a tree structure that includes public (standard) and private (proprietary) branches.

N

- navigation tree** Access to all CiscoWorks tasks and operations takes place through the navigation tree. Located in the left frame of the CiscoWorks window below the button bar, the navigation tree consists of multiple folders, each of which contains a group of associated or similar tasks, tools, or other options. The buttons in the button bar determine the contents of this tree.
- NMIM** Network Management Integration Module. Depending on the specific NMS, this utility can launch Cisco network management applications, browse Cisco MIBs, integrate traps, and add Cisco device icons to NMS topology maps. This utility also allows remote integration between CiscoWorks applications residing on one server and an SNMP management platform residing on another server. Also known as the Integration Utility.

O

- observable persistent object** Persistent objects that can be observed on an instance-by-instance basis. ObservablePO is a base class for observable persistent objects. See also [discovery](#).
- ODBC** Open Database Connectivity. A generic vendor independent API for accessing relational databases.
- ORB** Object Request Broker. Part of the Object Management Group (OMG) standard.

CISCO CONFIDENTIAL

P

- persistent object** PersistentObject is the base class for all objects that should be mirrored to the relational database in ANI. The value of a persistent object is isolated from other transactions, so results of bind() or isPersistent() calls do not become visible to queries or other transactions until the modifying transaction commits. When ANI is restarted, the persistent objects are read from the database. See also discovery.
- polling** Access method in which a primary network device inquires, in an orderly fashion, whether secondaries have data to transmit. The inquiry occurs in the form of a message to each secondary that gives the secondary the right to transmit.
- precondition** A function that must complete before another function can start. See also discovery.
- Process Manager** See [Daemon Manager](#).
- protocol** Formal description of a set of rules and conventions that govern how devices on a network exchange information.
- protopackage** A tar file that contains a component of the product. This component has a name, version, and other properties.

R

- router** Network layer device that uses one or more metrics to determine the optimal path along which network traffic should be forwarded. Occasionally called a gateway (although this definition of gateway is becoming increasingly outdated).

S

- seed device** A Cisco network device, such as a switch, that ANI uses to initiate discovery.
- service bundle** A collection of CWCS services. See also [CWCS Base Services](#), [CWCS Network Services](#), and [CWCS System Services](#).
- service module** Backend process components of the ANI Server that perform specific tasks to retrieve detailed information about network devices and topology.
- SMFContainer** Service module function container. Stores anything you want to do things to or collect things about.

For example, an SMFContainer stores the instances of functions for each device in memory for the current run of ANI. One SMFContainer is created for each evaluation thread.
- SMFFactory** Defines which functions are overridden by subclasses for each device type, overriding the default SMFunction with the new subclass. An SMFFactory instance is instantiated and configured for each device family type. See also discovery.
- SMFunction** Service module functions define the evaluation steps of a task. The sequence in which these functions are run is determined by the timebase. See also discovery.

CISCO CONFIDENTIAL

| | |
|-------------------------------|--|
| SNMP | Simple Network Management Protocol. Network management protocol used almost exclusively in TCP/IP networks. SNMP provides a means to monitor and control network devices and to manage configurations, statistics collection, performance, and security. |
| SNMP community strings | See community strings. |
| SQL | Structured Query Language. International standard language for defining and accessing relational databases. |
| SWIM | Software Management, a CiscoWorks application used to automate many steps associated with upgrade planning, scheduling, downloading, and monitoring software for managed devices on a network. |

T

| | |
|------------------|--|
| TCP/IP | Transmission Control Protocol/Internet Protocol. Common name for the suite of protocols developed by the U.S. DOD in the 1970s to support the construction of worldwide internetworks. TCP and IP are the two best-known protocols in the suite. |
| timebase | A named sequence of SMFunctions. See also discovery. |
| timeline | A timeline determines how often and when a timebase will be run. See also discovery. |
| timestamp | A field that records the time an event occurs. |
| topology | Physical arrangement of network nodes and media within an enterprise networking structure. |

V

| | |
|-------------|---|
| VLAN | virtual LAN. Group of devices on one or more LANs that are configured (using management software) so that they can communicate as if they were attached to the same wire, when in fact they are located on a number of different LAN segments. Because VLANs are based on logical instead of physical connections, they are extremely flexible. |
| VTP | Virtual Terminal Protocol. ISO application for establishing a virtual terminal connection across a network. |

W

| | |
|-------------------|---|
| web server | A computer that delivers web pages to your browser. |
|-------------------|---|

CISCO CONFIDENTIAL



CISCO CONFIDENTIAL

INDEX

A

ACLM, definition [1-21](#)

adding

 applications

 to the desktop [5-8](#)

 help engine, calls to [16-14](#)

 help page contents

 in default order [16-24](#)

 in specific order [16-25](#)

 online help [16-1](#)

 search support [16-27](#)

 third-party help systems [16-29](#)

 XML files [10-17](#)

Aelfred XML parser, third-party tool available with CMF [1-11](#)

aggregate table, definition [1-21](#)

ANI (Asynchronous Network Interface)

 definition [1-21](#)

AniAggregateTable, definition [1-21](#)

ANI Server

 definition [1-21](#)

APIs

 changes for this release [11-2](#)

 database (see database APIs) [11-1](#)

 definition [1-21](#)

 Helper (see Helper API methods in JRM) [18-49](#)

 Java Plug-in (see Java APIs) [22-2](#)

 JMS (see JMS API) [19-18](#)

 LWMS (see Lightweight Messaging Service) [19-15](#)

 Perl (see Perl APIs) [11-46](#)

 Process Manager [17-3](#)

 Windows-specific [21-37](#)

 Customer Order Number:

application security

 APIs, using [10-17](#)

 backend Perl script [10-17](#)

 HTML pages [10-17](#)

 Java applets [10-17](#)

 Java Server Pages (JSP) [10-18](#)

 Java servlets [10-17](#)

approve, JRM command [18-59](#)

ASA60, definition [1-21](#)

ATM

 definition [1-21](#)

 fabric, definition [1-21](#)

audience for this document [1-33](#)

authoring tool for online help, selecting [16-19](#)

 development environment, setting up [16-19](#)

 FrameMaker/WebWorks [16-21](#)

 RoboHELP [16-19](#)

auto login pages, creating [10-18](#)

B

backing up

 the Sybase database [11-32](#)

BACKUP, common runtime directory, path and contents [3-2](#)

BIN, common runtime directory, path and contents [3-2](#)

buttons

 context-sensitive help, linking [16-4](#)

buttons in

 Job Browser, customizing [18-22](#)

 behavior [18-22](#)

CISCO CONFIDENTIAL**C**

- C++
 - CSCOVorb for Visibroker and, package description [1-14](#)
 - interface in Process Manager, using [17-4](#)
 - Process Manager command summary (table) [17-11](#)
- cancel, JRM command [18-60](#)
- casuser
 - understanding and implementing [21-21](#)
 - non-CMF application, upgrading [21-21](#)
 - ownership for package files (see package file ownership) [21-86](#)
- cautions
 - CMF 2.1, ensuring your code works with [1-4](#)
 - configureDb utility [11-16](#)
 - data file deletion data files, caution regarding deleting [21-25](#)
 - Dependency Handler, using [21-27](#)
 - EncryptedObject encryption [10-19](#)
 - kill command, using [11-28](#)
 - package removal [21-25](#)
 - significance of [1-34](#)
 - subdirectory sharing [16-22](#)
 - system services, starting, in enabling new service bundles [5-5](#)
 - system services, starting, in using CMFEnable [5-6](#)
 - tools marked obsolete and deprecated [1-10](#)
 - Windows global values, and CMF [21-35](#)
 - Windows Task Manager, using [11-27](#)
- CCO, definition [1-21](#)
- CD image structure
 - Solaris [1-10](#)
 - Windows 2000/NT [1-10](#)
- CDP, definition [1-21](#)
- CEB (Client Event Bus), implemented by LWMS [19-13](#)
- CGI-BIN, common runtime directory, path and contents [3-2](#)
- changes for this release [11-2](#)
- client, definition [1-22](#)
- client/server, definition [1-22](#)
- CLIENTINSTALL, common runtime directory, path and contents [3-2](#)
- CLIENT-JAVA, common runtime directory, path and contents [3-2](#)
- CLIENT-JAVA-APPS, common runtime directory, path and contents [3-2](#)
- CMF
 - (see also CMF database, understanding) [11-2](#)
 - (see also CMF directory structure, understanding) [3-1](#)
 - adding your application to the desktop [5-8](#)
 - back-end services, using [5-8](#)
 - support tools, using [5-9](#)
 - benefits of using [1-3](#)
 - daemons, registering and unregistering [21-51](#)
 - definition [1-22](#)
 - desktop (see desktop) [6-1, 7-1, 8-1, 9-1, 14-1, 24-1, 25-1, 26-1, 29-1, 32-1, 33-1, 34-1](#)
 - directory structure (see CMF directory structure) [3-1](#)
 - distribution method [1-9](#)
 - CD image structure [1-9](#)
 - CMF base [1-9](#)
 - installation interface options [1-9](#)
 - execution environment, understanding [4-1](#)
 - cwjava command line options (table) [4-3](#)
 - Java application launch process, understanding [4-1](#)
 - Java applications, launching [4-2](#)
 - JavaBeans, using in CMF [4-5](#)
 - getting started with [5-1](#)
 - how it works [5-1](#)
 - installing CMF (see installing CMF) [5-2](#)
 - user interface, designing [5-7](#)
 - installation interface options [1-9](#)
 - CiscoView [1-9](#)
 - CMF Base Desktop [1-9](#)
 - NMS Integration Utility [1-9](#)
 - typical [1-9](#)
 - interacting with [5-7](#)

CISCO CONFIDENTIAL

- introduction to [1-1](#)
- licenses (see License APIs) [11-37](#)
- new or changed in this release [1-3](#)
- Perl APIs, using (see Perl APIs) [11-46](#)
- release 2.1, caution regarding your code and [1-4](#)
- release model [1-1](#)
- services, enabling [5-3](#)
 - CMFEnable, using [5-5](#)
 - new [5-5](#)
 - registering for [5-4](#)
 - service bundles, understanding [5-3](#)
- service subsets, descriptions of [1-7](#)
 - CORBA infrastructure [1-7](#)
 - database engine and utilities [1-8](#)
 - dbreader [1-8](#)
 - event management [1-8](#)
 - hidden tools [1-8](#)
 - install framework and patching process [1-8](#)
 - Java Plug-in [1-8](#)
 - Java Runtime Environment (JRE) [1-8](#)
 - Java SNMP engine [1-8](#)
 - job and resource management [1-8](#)
 - log file viewer [1-8](#)
 - online help system [1-8](#)
 - Perl interpreter [1-8](#)
 - process management [1-7](#)
 - security [1-8](#)
 - Web server and servlet engine [1-9](#)
- SSL, reasons for using in [10-20](#)
- SSL support
 - applications, enabling to work over SSL [10-21](#)
 - reasons for using SSL [10-20](#)
 - support types available [10-21](#)
- structure [1-5](#)
- third-party tools, and [1-10](#)
 - Aelfred XML parser (CSCOxrts) [1-11](#)
 - cautions regarding [1-10](#)
 - CSCOxsl [1-13](#)
 - Java 2 Runtime Environment (CSCOjre2) [1-12](#)
 - Java XML parser (CSCOsm14J) [1-15](#)
 - JChart (Jchart) [1-12](#)
 - Perl (CSCOperl) [1-14](#)
 - Visibroker for Java and C++ (CSCOvorb) [1-14](#)
 - Web Server (CSCOweb) [1-11](#)
- tools, understanding
 - DB Internal [11-2](#)
 - jConnect [11-2](#)
 - JDBC [11-2](#)
 - ODBC [11-2](#)
 - SQL database engine [11-2](#)
- CMF Base Services, definition [1-22](#)
- CMF database, understanding [11-2](#)
 - database access applications [11-3](#)
 - connection strings [11-5](#)
 - database server types [11-3](#)
 - JDBC access methods [11-4](#)
 - ODBC access methods [11-4](#)
 - Perl access methods [11-4](#)
- CMF directory structure, understanding [3-1](#)
 - file permissions [3-4](#)
 - policies [3-1](#)
 - top-level runtime directories [3-1](#)
 - common directories (table) [3-2](#)
 - Solaris-specific directories (table) [3-3](#)
 - UNIX-specific directories (table) [3-3](#)
 - Windows and UNIX, differences between [3-2](#)
 - Windows-specific directories (table) [3-3](#)
- CMFEnable, using [5-5](#)
- CMF Network Services, definition [1-22](#)
- CMF System Services, definition [1-22](#)
- CMF upgrade mechanisms
 - API [21-23](#)
 - for package upgrade, CopyOut API [21-24](#)
 - function prototypes [21-26](#)
 - Solaris example [21-26](#)
 - Windows 2000 and Windows NT example [21-26](#)
 - uninstallation before upgrade [21-25](#)

CISCO CONFIDENTIAL

COLLECT, common runtime directory, path and contents [3-2](#)

Common APIs

Adding Unauthenticated URLs [21-26](#)

common directories (table) [3-2](#)

BACKUP [3-2](#)

BIN [3-2](#)

CGI-BIN [3-2](#)

CLIENTINSTALL [3-2](#)

CLIENT-JAVA [3-2](#)

CLIENT-JAVA-APPS [3-2](#)

COLLECT [3-2](#)

CONF [3-2](#)

DATABASES [3-2](#)

DBMS [3-2](#)

DBUPGRADE [3-2](#)

ETC [3-2](#)

HELP [3-2](#)

HTDOCS [3-2](#)

HTDOCS-IMAGES [3-2](#)

LIB [3-2](#)

OBJECTS [3-2](#)

OBJECTS-APPS [3-2](#)

SELFTEST [3-2](#)

SERVER-JAVA [3-2](#)

SERVER-JAVA-APPS [3-3](#)

SETUP [3-3](#)

community strings

definition [1-22](#)

SNMP, definition [1-27](#)

CONF, common runtime directory, path and contents [3-2](#)

configureDb utility, caution regarding using [11-16](#)

CORBA

definition [1-22](#)

infrastructure, CWCS services [1-7](#)

create, JRM command [18-60](#)

CreateReadyFile, Process Manager command [17-18](#)

CSV, definition [1-22](#)

customizing

the message window

message directory files (table) [7-9](#)

cwjava

command

line options (table) [4-3](#)

JRE2 options (table) [4-4](#)

D

Daemon Manager

definition [1-22](#)

daemons

definition [1-22](#)

process, definition [1-22](#)

database API command reference

(see also database APIs) [11-1](#)

CMF Perl APIs, using (see Perl APIs) [11-46](#)

code samples

Java, using to read a database [11-38](#)

ODBC, using to access a table [11-40](#)

Perl, using to access a database [11-40](#)

database utilities, using

backup.pl [12-10](#)

configureDb.pl [11-53](#)

dbinit [11-54](#)

dbmonitor [11-55](#)

dbpasswd.pl [11-57](#)

dbreader.pl [11-59](#)

dbRestoreOrig [11-59](#)

dbvalid [11-60](#)

restorebackup.pl [12-11](#)

runIsqI [11-60](#)

JDBC API wrappers, using

Class DBUtil [11-43](#)

DBClient [11-41](#)

DBConnection [11-44](#)

DBResult [11-42](#)

database APIs [11-1](#)

(see also database API command reference) [11-37](#)

CISCO CONFIDENTIAL

- changes for this release [11-2](#)
- CMF database, understanding [11-2](#)
 - database access applications (figure) [11-3](#)
 - database access methods [11-3](#)
 - tools [11-2](#)
- command reference (see database API command reference) [11-37](#)
- database processes and settings [11-6](#)
 - backup manifest files, creating [11-10](#)
 - database engine, managing [11-13](#)
 - property files and settings [11-12](#)
 - template file, creating [11-7](#)
- debugging and troubleshooting databases [11-33](#)
 - abandoning [11-35, 12-21](#)
 - database log files, managing [11-33](#)
 - verifying [11-34](#)
- EMBU database delivery process, understanding [11-5](#)
- SqlAnywhere [11-30](#)
- SQLSecureConnect [11-28](#)
- Sybase database [11-18](#)
 - backing up [11-32](#)
 - connections, creating and closing [11-28](#)
 - contents, examining [11-31](#)
 - creating [11-20](#)
 - engines, starting and stopping [11-25](#)
 - environment, setting up [11-19](#)
 - initializing a new [11-19](#)
 - password, updating [11-24](#)
 - preparation for [11-19](#)
- database connections
 - C, using SQLDriverConnect [11-28](#)
 - C, using SQLSecureConnect [11-28](#)
 - Java, loading the JDBC driver [11-28](#)
 - Perl, using SqlAnywhere [11-30](#)
- database engine and utilities, CWCS services [1-8](#)
- database processes and settings
 - backup manifest files, creating [11-10](#)
 - database engine, managing
 - configureDb utility, caution regarding [11-16](#)
 - database port ID, changing [11-16](#)
 - database port ID, creating [11-15](#)
 - database port ID, dynamically allocating [11-17](#)
 - database port IDs, understanding [11-14](#)
 - database property files and settings
 - database server property file [11-12](#)
 - private property files [11-13](#)
 - database template file, creating
 - database password encryption, enabling [11-9](#)
 - odbc.ini template, creating [11-7](#)
 - odbc.tmplorig template, creating [11-7](#)
- DATABASES, common runtime directory, path and contents [3-2](#)
- DBClient constructors and public methods [11-41](#)
- DBConnection constructors and public methods [11-44](#)
- DBD, definition [1-22](#)
- DBI, definition [1-23](#)
- DB Internal, CMF database tool, description [11-2](#)
- DBMS, common runtime directory, path and contents [3-2](#)
- dbreader
 - accessing the data [11-32](#)
 - API, using [11-59](#)
 - creating a DSN [11-31](#)
- dbreader, CMF service subset [1-8](#)
- DBResult constructors and public methods [11-42](#)
- DBUPGRADE, common runtime directory, path and contents [3-2](#)
- DBUtil public methods [11-43](#)
- delay, JRM command [18-60](#)
- delete, JRM command [18-61](#)
- dependency handler, caution regarding using [21-27](#)
- deprecated and obsolete tools, caution regarding using [1-10](#)
- desktop, integrating an application with [6-1, 7-1, 8-1, 9-1, 14-1, 24-1, 25-1, 26-1, 29-1, 32-1, 33-1, 34-1](#)
 - desktop, using [7-1, 8-1, 9-1, 14-2, 24-1, 25-1, 32-6, 33-3, 34-4](#)
 - procedure [7-6](#)
 - security issues [10-16](#)
 - APIs, using [10-17](#)

CISCO CONFIDENTIAL

auto login pages, creating [10-18](#)

device

- adaptor, definition [1-23](#)
- class, definition [1-23](#)
- conflict, definition [1-23](#)

discovery

- definition [1-23](#)

DLL, definition [1-23](#)

DMCONFIG runtime directories, paths, and content

- Solaris-specific [3-3](#)

dMgtClose, Process Manager command [17-11](#)

dMgtCreateReadyFile, Process Manager command [17-12](#)

dMgtErr, Process Manager command [17-12](#)

dMgtGetMsg, Process Manager command [17-13](#)

dMgtInit, Process Manager command [17-13](#)

dMgtIsShutdown, Process Manager command [17-14](#)

dMgtProcessMsg, Process Manager command [17-14](#)

dMgtSendStatus, Process Manager command [17-14](#)

DMSTARTUP runtime directories, paths, and content

- Solaris-specific [3-3](#)

DNS, definition [1-23](#)

documentation

- new and updated pages [1-5](#)
- organization [1-35](#)
- related [1-34 to ??](#)

DSN, definition [1-23](#)

DTD

- definition [1-23](#)

E

eavesdropping, using SSL to stop [10-20](#)

EDS

- definition [1-23](#)
- using to publish events in JRM [20-4, 20-18](#)

EDS, see Event Distribution System [6-13, 19-1](#)

ELAN, definition [1-23](#)

EMBU database delivery process, understanding [11-5](#)

EncryptedObject encryption, caution regarding [10-19](#)

encryption, using CMF and Java APIs

- asymmetrical encryption
 - code sample [10-20](#)
 - when to use [10-20](#)
- symmetrical encryption
 - caution regarding [10-19](#)
 - code sample [10-19](#)
 - when to use [10-19](#)

ENG_42263 [16-1](#)

ENG-104742 [16-1](#)

ENG-116280 [16-1](#)

ENG-123901 [10-21](#)

ENG-21240 [17-1, 17-2](#)

ENG-29971

- CMF, installing [5-2](#)
- override dependency handler [21-27](#)

ENG-29984 [21-22](#)

ENG-54964 [11-1](#)

ENG-58367 [19-8, 19-12](#)

ENG-7045 [16-1](#)

ENG-72595 [19-8](#)

ENG-80264 [11-1](#)

enum_job_locks, JRM lock manager method [18-44](#)

Essentials, definition [1-23](#)

ETC, common runtime directory, path and contents [3-2](#)

EvalGroup, definition [1-23](#)

Event Distribution System [6-13, 19-1](#)

event management, CWCS services [1-8](#)

event services [19-1](#)

F

fabric, definition [1-24](#)

FAQs and programming hints [2-1](#)

- ANI [2-4](#)
- general topics [2-1](#)
- online help [2-5](#)
- Process Manager [2-4](#)

FILES runtime directories, paths, and content

CISCO CONFIDENTIAL

UNIX-specific [3-3](#)
 Windows-specific [3-3](#)
 filtering messages for a mailbox
 using the JMS API [19-19](#)
 using the LWMS API [19-16](#)
 find_lock, JRM lock manager method [18-45](#)
 FrameMaker
 online help authoring [16-21](#)
 setting up [16-21](#)
 single-source authoring of online help [16-19](#)
 framework, definition [1-24](#)

G

gateway, LWMS and Tibco [19-13](#)
 get_job_id, Helper API method in JRM [18-51](#)
 get_job_info, Helper API method in JRM [18-52](#)
 get_lock, JRM lock manager method [18-45](#)
 get_lock_info, Helper API method in JRM [18-52](#)
 GetCmdType, Process Manager command [17-18](#)
 GetConFile, Process Manager command [17-15](#)
 GetDescriptor, Process Manager command [17-15, 17-18](#)
 GetDmgtHostAndPort, Process Manager command [17-16](#)
 GetErr, Process Manager command [17-19](#)
 GetMsg, Process Manager command [17-19](#)
 getOrbConnectionProperties, Helper API method in JRM [18-53](#)
 getScheduleString, Helper API method in JRM [18-53](#)
 GetServerinfo, Process Manager command [17-19](#)
 getStateStrings, Helper API method in JRM [18-53](#)
 GetStatusMsg, Process Manager command [17-20](#)
 GUI
 definition [1-24](#)
 designing [5-7](#)

H

HELP, common runtime directory, path and contents [3-2](#)
 Helper API methods in JRM [18-49](#)

get_job_id [18-51](#)
 get_job_info [18-52](#)
 get_lock_info [18-52](#)
 GetOrbConnectionProperties [18-53](#)
 getScheduleString [18-53](#)
 getStateStrings [18-53](#)
 lock [18-54](#)
 lock_n [18-55](#)
 set_completion_state [18-55](#)
 set_progress [18-55](#)
 unlock [18-56](#)
 unlock_all [18-56](#)
 hidden tools, CMF service subset [1-8](#)
 HTDOCS, common runtime directory, path and contents [3-2](#)
 HTDOCS-IMAGES, common runtime directory, path and contents [3-2](#)
 HTML
 pages, in application security [10-17](#)
 HTTP, definition [1-24](#)

I

IDL (Interface Definition Language)
 definition [1-24](#)
 JRM server and [18-8](#)
 ILMI, definition [1-24](#)
 installable unit, definition [1-24](#)
 installation framework and patching process [21-1](#)
 build tools, getting started with [21-103](#)
 debugging, overview of [21-72, 21-105](#)
 framework, installing [21-70, 21-103](#)
 NT getting started example [21-72](#)
 protopackages, preparing [21-103](#)
 Solaris getting started examples [21-106](#)
 third-party tools, installing [21-70, 21-103](#)
 casuser, understanding and implementing [21-21](#)
 non-CMF application, upgrading [21-21](#)

CISCO CONFIDENTIAL

- ownership for package files (see package file ownership) [21-86](#)
- database upgrades [21-22](#)
 - paths and strategies [21-22](#)
 - RME upgrade approaches (figure) [21-23](#)
- dependency handler, overriding [21-27](#)
- including files in protopackage [21-20](#)
- licensing [21-22](#)
- main framework
 - developer responsibilities [21-3](#)
 - installation team responsibilities [21-3](#)
- patch CD, making [21-28](#)
- patching [21-27](#)
 - patches, creating [21-27](#)
 - patch mode installation [21-28](#)
 - policy [21-27](#)
- procedure
 - name value pairs in .pkgpr files (table) [21-7](#)
 - package name requirements (table) [21-6](#)
 - package names, selecting [21-6](#)
 - package properties, specifying [21-6, 21-7, 21-10](#)
 - properties appended during the build process (table) [21-9](#)
 - properties generated by the installer (table) [21-9](#)
 - protopackage file, required [21-20](#)
- scripts, writing for your package [21-33, 21-90](#)
 - Solaris hook types (see Solaris hook types) [21-90](#)
 - Windows hook types [21-33](#)
 - Windows-specific APIs [21-37](#)
- tables of contents [21-11](#)
 - file components of (table) [21-13, 21-16, 21-18](#)
 - properties, how the installer processes [21-18](#)
 - properties, specifying [21-19](#)
 - Solaris-specific properties, specifying [21-9](#)
- third-party tools [21-4](#)
- install framework and patching process, CMF service subset [1-8](#)
- INSTLOGS, UNIX-specific runtime directory, path and content [3-3](#)
- integrating your application with the desktop [7-6](#)

- Integration Utility, definition [1-24](#)
- Interface Definition Language (IDL), and the JRM server [18-8](#)
- IP address, definition [1-24](#)
- is_server_running, Helper API method in JRM [18-54](#)
- IsShutdownRequest, Process Manager command [17-20](#)

J

- Java
 - (see also Java Plug-in) [22-1](#)
 - (see also JDBC) [11-2](#)
 - applets, in application security [10-17](#)
 - applications
 - launching [4-2](#)
 - launch process, understanding [4-1](#)
 - database, reading with [11-38](#)
 - Java 2 Runtime Environment
 - description [1-12](#)
 - JavaBeans
 - using in CMF [4-5](#)
 - Java Runtime Environment, CWCS services [1-8](#)
 - Java Server Pages (JSP), in application security [10-18](#)
 - Process Manager
 - interface to, using [17-4](#)
 - methods in (see Process Manager command reference) [17-17](#)
 - servlets
 - in application security [10-17](#)
 - SNMP engine, CWCS services [1-8](#)
 - XML parser, third-party tool available with CMF, description [1-15](#)
- Java APIs, encryption using
 - asymmetrical
 - code sample [10-20](#)
 - when to use [10-20](#)
 - symmetrical
 - code sample [10-19](#)
 - when to use [10-19](#)

CISCO CONFIDENTIAL

- Java Plug-in [22-1](#)
 - API [22-2](#)
 - CWCS services [1-8](#)
 - external references [22-4](#)
 - how it works
 - known issues [22-2](#)
 - requirements [22-1](#)
 - known issues [22-2](#)
- JChart, third-party tool available with CMF, description [1-12](#)
- jConnect
 - definition [1-24](#)
 - description [11-2](#)
- JDBC (Java Database Connectivity) [11-2](#)
 - API wrappers, using [11-41](#)
 - DBClient [11-41](#)
 - DBConnection [11-44](#)
 - DBResult [11-42](#)
 - DBUtil [11-43](#)
 - description [11-2](#)
- JMS message selectors, and LWMS [19-19](#)
- JNI, definition [1-24](#)
- job_cancel, Job Manager method [18-31, 18-32](#)
- job_create, Job Manager method [18-33](#)
- job_delete, Job Manager method [18-34](#)
- job_enum, Job Manager method [18-35](#)
- job_get_info, Job Manager method [18-36](#)
- job_get_result, Job Manager method [18-37](#)
- job_get_schedule, Job Manager method [18-37](#)
- job_get_schedule_string, Job Manager method [18-38](#)
- job_run, Job Manager method [18-38](#)
- job_set_approved, Job Manager method [18-39](#)
- job_set_info, Job Manager method [18-39, 18-40](#)
- job_set_progress_string, Job Manager method [18-40](#)
- job_set_reference, Job Manager method [18-40](#)
- job_set_result, Job Manager method [18-41](#)
- job_set_resume, Job Manager method [18-41](#)
- job_set_schedule, Job Manager method [18-42](#)
- job and resource management, CWCS services [1-8](#)
- Job Browser
 - buttons, customizing [18-22](#)
 - customizing [18-22](#)
 - button behavior [18-22](#)
 - understanding [18-10](#)
- JRE, definition [1-24](#)
- JRM (Job and Resource Manager) [18-1, 20-1](#)
 - (see also JRM command reference) [18-26](#)
 - architecture
 - overview [18-5](#)
 - understanding [18-5](#)
 - command reference (see JRM command reference) [18-26](#)
 - definition [1-24](#)
 - EDS, using to publish events [20-4, 20-18](#)
 - event types [20-19](#)
 - registering [20-20](#)
 - severity codes [20-19](#)
 - enabling [18-12](#)
 - Java application, using JRM from [18-12](#)
 - connection, establishing [18-12](#)
 - crashed jobs, handling [18-18](#)
 - devices, accessing a locked [18-20](#)
 - devices, locking and unlocking [18-19](#)
 - disabled jobs, enabling [18-17](#)
 - job, creating [18-14](#)
 - job descriptions, obtaining [18-16](#)
 - job status, setting [18-15](#)
 - unapproved jobs, handling [18-16](#)
 - unavailable resource, handling [18-19](#)
- Job Browser (see Job Browser) [18-22](#)
- JRM server, understanding [18-7](#)
 - Helper API [18-9](#)
 - IDL interface [18-8](#)
 - jobs and resources [18-7](#)
 - JRM and other CMF processes, relationship [18-11](#)
 - JRM events [18-9](#)
 - JRM server classes [18-8](#)

CISCO CONFIDENTIAL

- services, understanding [18-2](#)
 - job scheduling [18-3](#)
 - locking parts of a device [18-5](#)
 - locking resources [18-4](#)
 - locking resources from another application [18-4](#)
 - managing [18-3](#)
- web browser, using JRM from a [18-21](#)
- JRM command reference [18-26](#)
 - command-line usage [18-24](#)
 - job CLI [18-25](#)
 - lock command line interface [18-25](#)
 - displayed jobs status values [18-28](#)
- Helper API methods [18-49](#)
 - get_job_id [18-51](#)
 - get_job_info [18-52](#)
 - get_lock_info [18-52](#)
 - getOrbConnectionProperties [18-53](#)
 - getScheduleString [18-53](#)
 - getStateStrings [18-53](#)
 - is_server_running [18-54](#)
 - lock [18-54](#)
 - lock_n [18-55](#)
 - set_completion_state [18-55](#)
 - set_progress [18-55](#)
 - unlock [18-56](#)
 - unlock_all [18-56](#)
- Java constants [18-56](#)
- job and resource lock attributes [18-26](#)
- job command line commands [18-59](#)
 - approve [18-59](#)
 - cancel [18-60](#)
 - create [18-60](#)
 - delay [18-60](#)
 - delete [18-61](#)
 - jobcli command summary (table) [18-59](#)
 - reject [18-61](#)
 - resume [18-61](#)
 - run [18-62](#)
 - schedule [18-62](#)
 - suspend [18-62](#)
- Job Manager methods [18-29](#)
 - job_cancel [18-31, 18-32](#)
 - job_create [18-33](#)
 - job_delete [18-34](#)
 - job_enum [18-35](#)
 - job_get_info [18-36](#)
 - job_get_result [18-37](#)
 - job_get_schedule [18-37](#)
 - job_get_schedule_string [18-38](#)
 - job_run [18-38](#)
 - job_set_approved [18-39](#)
 - job_set_info [18-39, 18-40](#)
 - job_set_progress_string [18-40](#)
 - job_set_reference [18-40](#)
 - job_set_result [18-41](#)
 - job_set_resume [18-41](#)
 - job_set_schedule [18-42](#)
 - next [18-42](#)
 - next_n [18-43](#)
 - release [18-43](#)
- lock manager methods [18-44](#)
 - enum_job_locks [18-44](#)
 - find_lock [18-45](#)
 - get_lock [18-45](#)
 - lock [18-46](#)
 - lock_n [18-46](#)
 - next [18-47](#)
 - next_n [18-47](#)
 - release [18-48](#)
 - unlock [18-48](#)
 - unlock_job [18-49](#)
 - unlock_n [18-49](#)

K

- kill command, caution regarding [11-28](#)

CISCO CONFIDENTIAL

L

- LANE, definition [1-25](#)
- LECS, definition [1-25](#)
- LES, definition [1-25](#)
- LIB, common runtime directory, path and contents [3-2](#)
- License APIs
 - database connection, using SQLSecureConnect [11-28](#)
 - license database, maintaining [11-37](#)
- License database, maintaining [11-37](#)
- lock
 - Helper API method in JRM [18-54](#)
 - JRM lock manager method [18-46](#)
- lock_n
 - Helper API method in JRM [18-55](#)
 - JRM lock manager method [18-46](#)
- log file viewer, CWCS services [1-8](#)
- LOG runtime directories, paths, and content
 - UNIX-specific [3-3](#)
 - Windows-specific [3-4](#)
- LWMS (Lightweight Messaging Service)
 - (see also LWMS command reference) [19-19](#)
 - API, using [19-15](#)
 - mailbox, posting a message to [19-16](#)
 - mailboxes, creating [19-15](#)
 - mailboxes, polling for new messages [19-16](#)
 - message listener, removing [19-16](#)
 - messages, filtering [19-16](#)
 - command reference [19-19](#)
 - LwmsClient public methods (table) [19-19](#)
 - LwmsMessage message creation methods (table) [19-20](#)
 - LwmsMsgEvent methods (table) [19-20](#)
 - LwmsMsgListener interface methods (table) [19-20](#)
 - native API messaging methods [19-19](#)
 - command reference (see LWMS command reference) [19-19](#)
 - configuring [19-13](#)
 - client properties [19-13](#)
 - server properties [19-14](#)
 - ENG-58367 [19-8](#)
 - ENG-72595 [19-8](#)
 - functional overview [19-10](#)
 - LwmsClient [19-11](#)
 - LwmsMailbox [19-11](#)
 - LwmsMessage [19-11](#)
 - LwmsServer [19-11](#)
 - JMS API, using [19-18](#)
 - mailbox, creating [19-18](#)
 - mailboxes, polling for new messages [19-18](#)
 - message, posting to a mailbox [19-18](#)
 - message listener, removing [19-18](#)
 - message selectors, working with [19-19](#)
 - JMS API support [19-12](#)
 - JMS to LWMS mappings [19-21](#)
 - JMS to LWMS API mappings (table) [19-21](#)
 - JMS to LWMS message field mappings (table) [19-21](#)
 - LWMS message queues [19-12](#)
 - LWMS server logging [19-12](#)
 - system usage assumptions [19-12](#)
 - Tibco-LWMS gateway support [19-13](#)
 - understanding [19-9](#)
 - components [19-9](#)
 - LWMS usage example (figure) [19-10](#)

M

- mailboxes
 - creating
 - using the JMS API [19-18](#)
 - using the LWMS API [19-15](#)
 - polling for new messages
 - using the JMS API [19-18](#)
 - using the LWMS API [19-16](#)
 - posting messages to
 - using the JMS API [19-18](#)
 - using the LWMS API [19-16](#)

CISCO CONFIDENTIAL

MAN, UNIX-specific runtime directory, path and content [3-3](#)

MDC, common runtime directory, path and contents [3-2](#)

message

- directory files (table) [7-9](#)
- filtering
 - using the JMS API [19-19](#)
 - using the LWMS API [19-16](#)
- listener, removing
 - using the JMS API [19-18](#)
 - using the LWMS API [19-16](#)
- polling for new
 - using the JMS API [19-18](#)
 - using the LWMS API [19-16](#)
- posting to a mailbox
 - using the JMS API [19-18](#)
 - using the LWMS API [19-16](#)
- queues, LWMS, understanding [19-12](#)
- selectors, working with in LWMS and JMS [19-19](#)
- window, customizing
 - message directory files (table) [7-9](#)

MIB, definition [1-25](#)

my_appdev [21-29](#)

- *.pkgpr, updating [21-29](#)
- CD, creating [21-30](#)
- my_app, patching with [21-31](#)
- protopackage, building the new [21-29](#)
- table ofcontents file, making [21-29](#)

N

names, creating subject [19-2](#)

navigation tree, definition [1-25](#)

navigation tree control, adding your application to

- procedures
 - XML file, adding [10-17](#)

next

- Job Manager method [18-42](#)
- JRM lock manager method [18-47](#)

next_n

- Job Manager method [18-43](#)
- JRM lock manager method [18-47](#)

NMIM, definition [1-25](#)

NMSROOT

- environment variable [3-1](#)

O

OBJECTS, common runtime directory, path and contents [3-2](#)

OBJECTS-APPS, common runtime directory, path and contents [3-2](#)

observable persistent object, definition [1-25](#)

obsolete and deprecated tools, caution regarding using [1-10](#)

ODBC (Open Database Connectivity)

- definition [1-25](#)
- description [11-2](#)
- device group tables, accessing with [11-40](#)

online help [16-1](#)

- (see also online help engineering tasks) [16-13](#)
- (see also online help mapping files) [16-9, 16-23](#)
- (see also online help search engine) [16-6](#)
- (see also online help writing tasks) [16-18](#)

appearance [16-2](#)

drop-in help systems, adding [16-29](#)

engine, understanding [16-4](#)

- context-sensitive help buttons, linking [16-4](#)
- display process, summary [16-5](#)
- error conditions, handling [16-5](#)
- help API process overview (figures) [16-6](#)
- help topics, displaying [16-4](#)
- main help contents and index, creating [16-5](#)

how it works [16-2](#)

online help system, CWCS services [1-8](#)

third-party systems, dropping in [16-29](#)

online help engineering tasks [16-13](#)

- help engine, call to, adding [16-14](#)

CISCO CONFIDENTIAL

- help files, packaging [16-16](#)
- help SDK files (client/web server), installing [16-13](#)
- mapping file, updating [16-15](#)
- required run-time help structure (figure) [16-16](#)
- online help mapping files [16-23](#)
 - creating [16-23](#)
 - main help page, defining contents and index for [16-24](#)
 - contents, adding in default order [16-24](#)
 - contents, adding in specific order [16-25](#)
 - figure [16-24](#)
 - search support, adding [16-27](#)
 - understanding [16-9](#)
 - conventions and requirements [16-10](#)
 - line type descriptions (table) [16-11](#)
 - line types [16-10](#)
 - sample mapping file [16-12](#)
- online help search engine, understanding [16-6](#)
 - Search dialog box
 - displaying [16-7](#)
 - figure [16-7](#)
 - search process, summary [16-8](#)
 - Search results output page (figure) [16-8](#)
- online help writing tasks [16-18](#)
 - authoring [16-22](#)
 - authoring tool, selecting [16-19](#)
 - FrameMaker/WebWorks [16-19](#)
 - delivering the help system [16-28](#)
 - development environments
 - FrameMaker/WebWorks [16-21](#)
 - RoboHELP [16-19](#)
 - setting up [16-19](#)
 - subdirectory sharing, caution regarding [16-22](#)
 - ENG-104742 [16-19](#)
 - ENG-70452 [16-19](#)
 - mapping file (see online help mapping files) [16-23](#)
 - search index file, maintaining [16-28](#)
- ORB, definition [1-25](#)

P

- package
 - file ownership, setting in Solaris [21-86](#)
 - during build or installation [21-88](#)
 - package_name.owner file, from [21-87](#)
 - package_name.owner file, understanding [21-88](#)
 - removal, caution regarding [21-25](#)
- packages
 - CSCOjchart (JChart), description [1-12](#)
 - CSCOjre2 (Java 2 Runtime Environment), description [1-12](#)
 - CSCOperl (Perl)
 - description [1-14](#)
 - CSCOsm4j (Java XML parser), description [1-15](#)
 - CSCOvorb (Visibroker for Java and C++), description [1-14](#)
 - CSCOWeb (Web Server), description [1-11](#)
 - CSCOxrts (Aelfred XML parser), description [1-11](#)
 - CSCOxsl (CSCOxIs), description [1-13](#)
- password encryption, enabling [11-9](#)
- password for a new database, warning regarding changing [11-20](#)
- patching process, CWCS services [1-8](#)
- pdexec, Process Manager command [17-6](#)
- pdrapp, Process Manager command [17-6](#)
- pdshow, Process Manager command [17-9](#)
- pdterm, Process Manager command [17-10](#)
- Perl
 - access methods [11-4](#)
 - APIs
 - addManifestFiles [11-47](#)
 - check_create [11-48](#)
 - checkDb [11-48](#)
 - database process management APIs [11-46](#)
 - deleteDbVersionData [11-48](#)
 - deleteManifestFiles [11-49](#)
 - getDbVersionData [11-49](#)
 - getManifestFiles [11-50](#)
 - miscellaneous APIs [11-46](#)

CISCO CONFIDENTIAL

- setDbVersionData [11-51](#)
 - StartDb [11-51](#)
 - StopDb [11-51](#)
 - unloadDbVersionData [11-52](#)
 - validateDb [11-52](#)
 - application security, backend script [10-17](#)
 - CSCOperl [1-14](#)
 - interpreter, CWCS services [1-8](#)
 - script, backend, in application security [10-17](#)
 - third-party tool available with CMF, description [1-14](#)
 - persistent object, definition [1-26](#)
 - polling, definition [1-26](#)
 - precondition, definition [1-26](#)
 - process management, CWCS services [1-7](#)
 - Process Manager [17-1](#)
 - (see also Process Manager command reference) [17-5](#)
 - API, using [17-3](#)
 - C++ interface, using [17-4](#)
 - command reference (see Process Manager command reference) [17-5](#)
 - commands (see Process Manager command reference) [17-5](#)
 - definition [1-26](#)
 - Java interface, using [17-4](#)
 - log files, writing messages to [17-5](#)
 - understanding [17-1](#)
 - using [17-2](#)
 - CLI [17-3](#)
 - starting and stopping [17-2](#)
 - Process Manager command reference [17-5](#)
 - ANSI C and C++ commands [17-11](#)
 - dMgtClose [17-11](#)
 - dMgtCreateReadyFile [17-12](#)
 - dMgtErr [17-12](#)
 - dMgtGetMsg [17-13](#)
 - dMgtInit [17-13](#)
 - dMgtIsShutdown [17-14](#)
 - dMgtProcessMsg [17-14](#)
 - dMgtSendStatus [17-14](#)
 - GetConFile [17-15](#)
 - GetDescriptor [17-15](#)
 - GetDmgtHostAndPort [17-16](#)
 - ValidatePgmPath [17-16](#)
 - command line utilities [17-6](#)
 - pdexec [17-6](#)
 - pdrapp [17-6](#)
 - pdregpdreg, Process Manager command [17-7](#)
 - pdshow [17-9](#)
 - pdterm [17-10](#)
 - Java methods [17-17](#)
 - CreateReadyFile [17-18](#)
 - GetCmdType [17-18](#)
 - GetDescriptor [17-18](#)
 - GetErr [17-19](#)
 - GetMsg [17-19](#)
 - GetServerInfo [17-19](#)
 - GetStatusMsg [17-20](#)
 - IsShutdownRequest [17-20](#)
 - ProcessMsg [17-20](#)
 - ReqStatus [17-21](#)
 - SendBusyMsg [17-21](#)
 - SendErrMsg [17-21](#)
 - SendOkMsg [17-22](#)
 - StartProcess [17-22](#)
 - Status [17-23](#)
 - StopProcess [17-22](#)
 - ProcessMsg, Process Manager command [17-20](#)
 - protocol, definition [1-26](#)
 - protopackage, definition [1-26](#)
 - PROXY, Windows-specific runtime directory, path and content [3-4](#)
-
- ## R
- reject, JRM command [18-61](#)
 - release
 - Job Manager method [18-43](#)
 - JRM lock manager method [18-48](#)

CISCO CONFIDENTIAL

- ReqStatus, Process Manager command [17-21](#)
 - resume, JRM command [18-61](#)
 - RoboHELP
 - writing environment [16-19](#)
 - router, definition [1-26](#)
 - run, JRM command [18-62](#)
-
- ### S
- schedule, JRM command [18-62](#)
 - Security
 - database password encryption, enabling [11-9](#)
 - security
 - certificates in CMF [10-21](#)
 - CWCS services [1-8](#)
 - security system, using [10-1](#)
 - desktop, integrating applications with [10-16](#)
 - APIs, using [10-17](#)
 - auto login pages, creating [10-18](#)
 - eavesdropping, using SSL to stop [10-20](#)
 - reasons for using SSL in CMF [10-20](#)
 - SSL support available in CMF [10-21](#)
 - encryption, using CMF and Java APIs
 - asymmetrical encryption [10-20](#)
 - symmetrical encryption [10-19](#)
 - server security [10-2](#)
 - general concerns [10-7](#)
 - seed device, definition [1-26](#)
 - self-signed certificates [10-21](#)
 - SELFTEST, common runtime directory, path and contents [3-2](#)
 - SendBusyMsg, Process Manager command [17-21](#)
 - SendErrMsg, Process Manager command [17-21](#)
 - SendOkMsg, Process Manager command [17-22](#)
 - server event bus (SEB) [19-2](#)
 - SERVER-JAVA, common runtime directory, path and contents [3-2](#)
 - SERVER-JAVA-APPS, common runtime directory, path and contents [3-3](#)
 - server logging, LWMS, understanding [19-12](#)
 - server security
 - general concerns [10-7](#)
 - service
 - bundles
 - definition [1-26](#)
 - understanding [5-3](#)
 - module, definition [1-26](#)
 - set_completion_state, Helper API method in JRM [18-55](#)
 - set_progress, Helper API method in JRM [18-55](#)
 - SETUP, common runtime directory, path and contents [3-3](#)
 - shared secret administration, understanding
 - shared secret client API
 - details of [10-11](#)
 - SecretClient.dumpResponse [10-14](#)
 - SecretClient.getErrCode [10-12](#)
 - SecretClient.secretLogon [10-12](#)
 - SecretClient.doPost [10-13](#)
 - SecretClient.getErrReason [10-13](#)
 - SMFContainer, definition [1-26](#)
 - SMFFactory, definition [1-26](#)
 - SMFunction, definition [1-26](#)
 - SNMP
 - community strings, definition [1-27](#)
 - definition [1-27](#)
 - Solaris
 - (see also Solaris hook types) [21-90](#)
 - debugging [21-105](#)
 - distribution CD image structure [1-10](#)
 - getting started example [21-106](#)
 - make image, preparing [21-103](#)
 - package file ownership, setting [21-86](#)
 - during build or installation [21-88](#)
 - package_name.owner, from [21-87](#)
 - package name owner, understanding [21-88](#)
 - Solaris-specific directories (table) [3-3](#)
 - DMCONFIG [3-3](#)
 - DMSTARTUP [3-3](#)

CISCO CONFIDENTIAL

- Solaris hook types [21-90](#)
 - Solaris-specific information
 - input/output APIs [21-92](#)
 - installable unit APIs [21-102](#)
 - package APIs [21-97](#)
 - system APIs [21-100](#)
 - SQL
 - Database Engine
 - description [11-2](#)
 - definition [1-27](#)
 - SSH (Secure Shell)
 - API, using
 - closing connection [10-15](#)
 - connecting to device [10-14](#)
 - debugging [10-15](#)
 - reading from device [10-15](#)
 - sending commands to a device [10-15](#)
 - SSL
 - eavesdropping, using to stop [10-20](#)
 - reasons for using in CMF [10-20](#)
 - support in CMF [10-21](#)
 - applications, enabling to work over SSL [10-21](#)
 - available [10-21](#)
 - StartProcess, Process Manager command [17-22](#)
 - Status, Process Manager command [17-23](#)
 - StopProcess, Process Manager command [17-22](#)
 - structural overview of CMF (figure) [1-6](#)
 - subdirectory sharing, caution regarding [16-22](#)
 - subject names, creating [19-2](#)
 - suspend, JRM command [18-62](#)
 - SWIM, definition [1-27](#)
 - Sybase database
 - backing up [11-32](#)
 - connections
 - closing [11-30](#)
 - creating [11-28](#)
 - contents, examining
 - database contents, accessing [11-32](#)
 - DSN, creating [11-31](#)
 - creating [11-20](#)
 - database files, installing [11-23](#)
 - DbVersion and DbVersionHistory, creating and populating [11-21](#)
 - user ID and password, changing [11-21](#)
 - engines
 - kill command, caution regarding [11-28](#)
 - starting [11-25](#)
 - stopping [11-27](#)
 - system services, caution regarding starting [5-5](#)
-
- T**
- TCP/IP, definition [1-27](#)
 - TEMP runtime directories, paths, and content
 - UNIX-specific [3-3](#)
 - Windows-specific [3-4](#)
 - TFTPBOOT, Windows-specific runtime directory, path and content [3-4](#)
 - third-party
 - help systems, adding [16-29](#)
 - tools, descriptions [1-10](#)
 - Aelfred XML parser (CSCOxrts), description [1-11](#)
 - caution regarding [1-10](#)
 - CSCOxsl [1-13](#)
 - Java 2 Runtime Environment (CSCOjre2) [1-12](#)
 - Java XML parser (CSCOxml4j) [1-15](#)
 - JChart (JChart) [1-12](#)
 - Perl (CSCOperl) [1-14](#)
 - Visibroker for Java and C++ (CSCOvorb) [1-14](#)
 - Web Server (CSCOweb) [1-11](#)
 - Tibco-LWMS gateway support [19-13](#)
 - Tibco Rendezvous [19-2](#)
 - tiers, and CMF service bundles [5-3](#)
 - timebase, definition [1-27](#)
 - timeline, definition [1-27](#)
 - timestamp
 - definition [1-27](#)

CISCO CONFIDENTIAL

tools marked obsolete and deprecated, caution regarding using [1-10](#)

topology

definition [1-27](#)

troubleshooting the database [11-33](#)

database log files, managing [11-33](#)

restoring a database

abandoning the database [11-35, 12-21](#)

from a previous backup [11-35, 12-19](#)

partial recovery [12-19](#)

verifying a database [11-34](#)

typographical conventions used in this document [1-33 to 1-34](#)

U

UNIX-specific runtime directories (table) [3-3](#)

FILES [3-3](#)

INSTLOGS [3-3](#)

LOG [3-3](#)

MAN [3-3](#)

TEMP [3-3](#)

unlock

Helper API method in JRM [18-56](#)

JRM lock manager method [18-48](#)

unlock_all, Helper API method in JRM [18-56](#)

unlock_job, JRM lock manager method [18-49](#)

unlock_n, JRM lock manager method [18-49](#)

userid for a new database, warning regarding changing [11-20](#)

V

ValidatePgmPath, Process Manager command [17-16](#)

Visibroker for Java and C++, third-party tool available with CMF [1-14](#)

VLAN, definition [1-27](#)

VTP, definition [1-27](#)

W

warning regarding changing database userid and password [11-20](#)

web server

definition [1-27](#)

servlet engine, and, CWCS services [1-9](#)

Web Server, third-party tool available with CMF, description [1-11](#)

WebWorks, online help authoring tool [16-21](#)

what's new for this release [1-3, 11-2](#)

Why SNMPv3? [24-1](#)

Windows 2000 and Windows NT

APIs [21-37](#)

distribution CD image structure [1-10](#)

getting started example [21-72](#)

global values and CMF, caution regarding [21-35](#)

hook types, writing [21-33](#)

installation APIs

CMF daemons, registering and unregistering [21-51](#)

commands, running in a shell [21-52](#)

generic utilities, using [21-58](#)

global variables, using [21-35](#)

informational messages, sending to a log file [21-48](#)

informing the installer of space needs [21-50](#)

NT services, registering and controlling [21-55](#)

package properties, and version comparisons [21-37](#)

processing files containing NAME=VALUE pairs [21-46](#)

root directory pathname, locating [21-54](#)

terminated installation, controlling a response to [21-45](#)

Winwows global values and CMF, caution regarding [21-35](#)

pkg.rul installation file [21-34](#)

runtime directories (table) [3-3](#)

FILES [3-3](#)

LOG [3-4](#)

CISCO CONFIDENTIAL

PROXY [3-4](#)

TEMP [3-4](#)

TFTPBOOT [3-4](#)

Windows Task Manager, caution regarding [11-27](#)

X

XML file

adding a new [10-17](#)