CHAPTER 25

Using NT Services

The CWCS NT Services (CSCOsvc) 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, TELNETand 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) l*
- *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.

- 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.

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
```

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.

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 writtenin 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 maximun 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 messgaes according to RFC 3164. The default value is 5.
QueueCapacity	Sets the maximum size of the internal queue. This queue contain the unprocessed syslog messages from UDP socket. The default value is 100000. If the CrmMsgCount is 256, then the queue can contain upto 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.

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 espeically 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 *suit*e, 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

- 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

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

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: • 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.

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.