

*InCharge*TM

XML Adapter User's Guide

Version 1.0



Copyright ©1996-2003 by System Management ARTS Incorporated. All rights reserved.

The Software and all intellectual property rights related thereto constitute trade secrets and proprietary data of SMARTS and any third party from whom SMARTS has received marketing rights, and nothing herein shall be construed to convey any title or ownership rights to you. Your right to copy the software and this documentation is limited by law. Making unauthorized copies, adaptations, or compilation works is prohibited and constitutes a punishable violation of the law. Use of the software is governed by its accompanying license agreement. The documentation is provided "as is" without warranty of any kind. In no event shall System Management ARTS Incorporated ("SMARTS") be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for indirect, special, incidental, or consequential damages of any kind, arising from any error in this documentation.

The InCharge products mentioned in this document are covered by one or more of the following U.S. patents or pending patent applications: 5,528,516, 5,661,668, 6,249,755, 10,124,881 and 60,284,860.

"InCharge," the InCharge logo, "SMARTS," the SMARTS logo, "Graphical Visualization," "Authentic Problem," "Codebook Correlation Technology," and "Instant Results Technology" are trademarks or registered trademarks of System Management ARTS Incorporated. All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Third-Party Software. The Software may include software of third parties from whom SMARTS has received marketing rights and is subject to some or all of the following additional terms and conditions:

Bundled Software

Sun Microsystems, Inc., Java(TM) Interface Classes, Java API for XML Parsing, Version 1.1. "Java" and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. SMARTS is independent of Sun Microsystems, Inc.

W3C IPR Software

Copyright © 2001-2003 World Wide Web Consortium (<http://www.w3.org>), (Massachusetts Institute of Technology (<http://www.lcs.mit.edu>), Institut National de Recherche en Informatique et en Automatique (<http://www.inria.fr>), Keio University (<http://www.keio.ac.jp>)). All rights reserved (<http://www.w3.org/Consortium/Legal/>). Note: The original version of the W3C Software Copyright Notice and License can be found at <http://www.w3.org/Consortium/Legal/copyright-software-19980720>.

The Apache Software License, Version 1.1

Copyright ©1999-2003 The Apache Software Foundation. All rights reserved. Redistribution and use of Apache source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of Apache source code must retain the above copyright notice, this list of conditions and the Apache disclaimer as written below.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the Apache disclaimer as written below in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:
"This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."
Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
4. The names "The Jakarta Project", "Tomcat", "Xalan", "Xerces", and "Apache Software Foundation" must not be used to endorse or promote products derived from Apache software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this Apache software may not be called "Apache," nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

APACHE DISCLAIMER: THIS APACHE SOFTWARE FOUNDATION SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA, OR PROFITS, OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This Apache software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright © 1999, Lotus Development Corporation., <http://www.lotus.com>. For information on the Apache Software Foundation, please see <http://www.apache.org>.

FLEXIm Software

© 1994 - 2003, Macrovision Corporation. All rights reserved. "FLEXIm" is a registered trademark of Macrovision Corporation. For product and legal information, see <http://www.macrovision.com/solutions/esd/flexim/flexim.shtml>.

JfreeChart – Java library for GIF generation

The Software is a "work that uses the library" as defined in GNU Lesser General Public License Version 2.1, February 1999 Copyright © 1991, 1999 Free Software Foundation, Inc., and is provided "AS IS" WITHOUT WARRANTY OF ANY KIND EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED IN THE ABOVE-REFERENCED LICENSE BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL,

INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. JfreeChart library (included herein as .jar files) is provided in accordance with, and its use is covered by the GNU Lesser General Public License Version 2.1, which is set forth at <http://www.object-refinery.com/lgpl.html/>.

BMC – product library

The Software contains technology (product library or libraries) owned by BMC Software, Inc. ("BMC Technology"). BMC Software, Inc., its affiliates and licensors (including SMARTS) hereby disclaim all representations, warranties and liability for the BMC Technology.

Crystal Decisions Products

The Software may contain certain software and related user documentation (e.g., Crystal Enterprise Professional, Crystal Reports Professional and/or Crystal Analysis Professional) that are owned by Crystal Decisions, Inc., 895 Emerson Street, Palo Alto, CA 94301 ("Crystal Decisions"). All such software products are the technology of Crystal Decisions. The use of all Crystal Decisions software products is subject to a separate license agreement included with the Software electronically, in written materials, or both. YOU MAY NOT USE THE CRYSTAL DECISIONS SOFTWARE UNLESS AND UNTIL YOU READ, ACKNOWLEDGE AND ACCEPT THE TERMS AND CONDITIONS OF THE CRYSTAL DECISIONS' SOFTWARE LICENSE AGREEMENT. IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS OF THE CRYSTAL DECISIONS' SOFTWARE LICENSE, YOU MAY RETURN, WITHIN THIRTY (30) DAYS OF PURCHASE, THE MEDIA PACKAGE AND ALL ACCOMPANYING ITEMS (INCLUDING WRITTEN MATERIALS AND BINDERS OR OTHER CONTAINERS) RELATED TO THE CRYSTAL DECISIONS' TECHNOLOGY, TO SMARTS FOR A FULL REFUND, OR YOU MAY WRITE, CRYSTAL WARRANTIES, P.O. BOX 67427, SCOTTS VALLEY, CA 95067, U.S.A.

GNU eTeks PJA Toolkit

Copyright © 2000-2001 Emmanuel PUYBARET/eTeks info@eteks.com. All Rights Reserved.

The eTeks PJA Toolkit is resident on the CD on which the Software was delivered to you. Additional information is available at eTEKS' web site: <http://www.eteks.com>. The eTeks PJA Toolkit program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation; version 2 of the License. The full text of the applicable GNU GPL is available for viewing at <http://www.gnu.org/copyleft/gpl.txt>. You may also request a copy of the GPL from the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. The eTeks PJA Toolkit program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY, without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

For a period of three years from the date of your license for the Software, you are entitled to receive under the terms of Sections 1 and 2 of the GPL, for a charge no more than SMARTS' cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code for the GNU eTeks PJA Toolkit provided to you hereunder by requesting such code from SMARTS in writing: Attn: Customer Support, SMARTS, 44 South Broadway, White Plains, New York 10601.

IBM Runtime for AIX

The Software contains the IBM Runtime Environment for AIX(R), Java™ 2 Technology Edition Runtime Modules © Copyright IBM Corporation 1999, 2000 All Rights Reserved.

HP-UX Runtime Environment for the Java™ 2 Platform

The Software contains the HP-UX Runtime for the Java™ 2 Platform, distributed pursuant to and governed by Hewlett-Packard Co. ("HP") software license terms set forth in detail at: <http://www.hp.com>. Please check the Software to determine the version of Java runtime distributed to you.

DataDirect Technologies

Portions of this software are copyrighted by DataDirect Technologies, 1991-2002.

NetBSD

Copyright (c) 2001 Christopher G. Demetriou. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
This product includes software developed for the NetBSD Project. See <http://www.netbsd.org/> for information about NetBSD.
4. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOSS OF USE, DATA, OR PROFITS, OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. <<Id: LICENSE, v 1.2 2000/06/14 15:57:33 cgd Exp>>

Contents

	iv
Preface	ix
Intended Audience	ix
Prerequisites	ix
Document Organization	x
Documentation Conventions	x
InCharge Installation Directory	xi
Additional Resources	xiii
InCharge Commands	xiii
Documentation	xiii
Technical Support	xiv
1 Introduction	1
Purpose of the XML Adapter	1
XML Adapter Process	2
Framework Description	3
ICIM/XML DTD Specification	3
ICIM/XML Document	4
Command Line Interface (sm_xml)	4
XML Export ASL Script	5
Structure of the XML Document	5
Element	5
Attribute Declaration (Attribute List)	5
Value	6
Rules for a Well-Formed XML Document	6

2	ICIM XML Elements	7
	Element Declarations	7
	Attribute-List Declarations	8
	Class	8
	Name	8
	Method	8
	Object Elements	9
	Attribute Elements	10
	Relationship Elements	12
	ICIM/XML DTD Specification	14
3	Building Topology in XML	15
	Using XML For InCharge	16
	Adding and Updating Topology	16
	Adding Objects	16
	Adding Attributes	17
	Adding Relationships Using the Put Method	19
	Adding Relationships Using the Insert Method	20
	Removing Topology	21
	Removing Objects	22
	Removing Attributes and Relationships	22
	Data Types	23
4	Exporting InCharge Information to XML	25
	The ASL Export Script	26
	Understanding the ASL Script	26
	Building the ASL Export Script	27
	getInstance Function	27
	Foreach Statement	27
	Builder Statements	28
	Parsing the XML	30
	Running sm_xml for Export	30
	Sample ASL Export Script	32

5	Importing Information Into InCharge	35
	XML Import Process	35
	Before Importing XML	36
	Running sm_xml to Import XML	37
	Index	39

Preface

Intended Audience

This document is intended for integrators using the InCharge XML Adapter to exchange topological information between InCharge and other applications.

Prerequisites

A version 6.0 or later InCharge Manager (such as Service Assurance Manager or IP Availability Manager) must be installed. You should have an understanding of the InCharge Common Information Model™ (ICIM) and XML concepts. In addition, if you plan to use the XML Adapter to export information from an InCharge Manager, you should have a strong understanding of the Adapter Scripting Language (ASL).

Document Organization

This guide consists of the following sections:

1. INTRODUCTION	Describes the purpose of the XML Adapter and the files that comprise the framework of the XML Adapter
2. ICIM XML ELEMENTS	Describes the elements that are used to build the XML document.
3. BUILDING TOPOLOGY IN XML	Describes how to create, update and delete topology objects using ICIM XML declarations.
4. EXPORTING INCHARGE INFORMATION TO XML	Describes how to build the export module and use the sm_xml command to export InCharge topology to an XML file.
5. IMPORTING INFORMATION INTO INCHARGE	Describes how to import an XML document into InCharge.

Table 1: Document Organization

Documentation Conventions

Several conventions may be used in this document as shown in Table 2.

CONVENTION	EXPLANATION
sample code	Indicates code fragments and examples in Courier font
keyword	Indicates commands, keywords, literals, and operators in bold
%	Indicates C shell prompt
#	Indicates C shell superuser prompt
<parameter>	Indicates a user-supplied value or a list of non-terminal items in angle brackets
[option]	Indicates optional terms in brackets
/InCharge	Indicates directory path names in italics
yourDomain	Indicates a user-specific or user-supplied value in bold, italics
File > Open	Indicates a menu path in italics
▲ ▼	Indicates a command that is formatted so that it wraps over one or more lines. The command must be typed as one line.

Table 2: Documentation Conventions

Directory path names are shown with forward slashes (/). Users of the Windows operating systems should substitute back slashes (\) for forward slashes.

Also, if there are figures illustrating consoles in this document, they represent the consoles as they appear in Windows. Under UNIX, the consoles appear with slight differences. For example, in views that display items in a tree hierarchy such as the Topology Browser, a plus sign displays for Windows and an open circle displays for UNIX.

Finally, unless otherwise specified, the term InCharge Manager is used to refer to InCharge programs such as Domain Managers, Global Managers, and adapters.

InCharge Installation Directory

In this document, the term **BASEDIR** represents the location where InCharge software is installed.

- For UNIX, this location is: `/opt/InCharge<n>/<productsuite>`.
- For Windows, this location is: `C:\InCharge<n>\<productsuite>`.

The `<n>` represents the InCharge software version number. The `<productsuite>` represents the InCharge product suite that the product is part of.

Table 3 defines the `<productsuite>` directory for each InCharge product.

PRODUCT SUITE	INCLUDES THESE PRODUCTS	DIRECTORY
IP Management Suite	<ul style="list-style-type: none"> • InCharge IP Availability Manager • InCharge IP Performance Manager • InCharge Discovery Manager • InCharge Adapter for HP OpenView NNM • InCharge Adapter for IBM/Tivoli NetView 	/IP
Service Assurance Management Suite	<ul style="list-style-type: none"> • InCharge Service Assurance Manager • Global Console • InCharge Service Assurance Manager Business Impact Manager • InCharge Service Assurance Manager Failover System • InCharge Service Assurance Manager Notification Adapters • InCharge Service Assurance Manager Adapter Platform • InCharge SNMP Trap Adapter • InCharge Syslog Adapter • InCharge XML Adapter • InCharge Adapter for Remedy • InCharge Adapter for TIBCO Rendezvous • InCharge Adapter for Concord eHealth • InCharge Adapter for InfoVista 	/SAM
Application Management Suite	<ul style="list-style-type: none"> • InCharge Application Connectivity Monitor 	/APP
SMARTS Software Development Kit	<ul style="list-style-type: none"> • Software Development Kit 	/SDK

Table 3: Product Suite Directory for InCharge Products

For example, on UNIX operating systems, version 6.0 of InCharge IP Availability Manager is, by default, installed to `/opt/InCharge6/IP/smarts`. This location is referred to as **BASEDIR**/`smarts`.

Optionally, you can specify the root of **BASEDIR** to be something other than `/opt/InCharge6` (on UNIX) or `C:\InCharge6` (on Windows), but you cannot change the `<productsuite>` location under the root directory.

For more information about the directory structure of InCharge software, refer to the *InCharge System Administration Guide*.

Additional Resources

In addition to this manual, SMARTS provides the following resources.

InCharge Commands

Descriptions of InCharge commands are available as HTML pages. The *index.html* file, which provides an index to the various commands, is located in the **BASEDIR**/*smarts/doc/html/usage* directory.

Documentation

Readers of this manual may find other SMARTS documentation (also available in the **BASEDIR**/*smarts/doc/pdf* directory) helpful.

InCharge Documentation

The following SMARTS documents are product independent and thus relevant to users of all InCharge products:

- *InCharge Release Notes*
- *InCharge Documentation Roadmap*
- *InCharge Installation Guide*
- *InCharge System Administration Guide*
- *InCharge Operator's Guide*

InCharge Service Assurance Manager Documentation

The following SMARTS documents are relevant to users of the InCharge Service Assurance Management product suite.

- *An Introduction to InCharge Service Assurance Manager*
- *InCharge Service Assurance Manager Configuration Guide*
- *InCharge Service Assurance Manager Failover System User's Guide*
- *InCharge Service Assurance Manager User's Guide for Business Impact Manager*

The following SMARTS documents are relevant to InCharge Service Assurance Manager adapters.

- *InCharge Service Assurance Manager Notification Adapters User's Guide*
- *InCharge Service Assurance Manager Adapter Platform User's Guide*

- *InCharge XML Adapter User's Guide*
- *InCharge Service Assurance Manager User's Guide for Remedy Adapter*
- *InCharge Service Assurance Manager User's Guide for Concord eHealth Adapter*
- *InCharge Service Assurance Manager User's Guide for InfoVista Adapter*

Technical Support

SMARTS provides technical support by e-mail or phone during normal business hours (9:00 A.M.—6:00 P.M. U.S. Eastern Time).

TECHNICAL SUPPORT: *support@smarts.com*

SALES: *sales@smarts.com*

WORLD WIDE WEB: *http://www.smarts.com*

TELEPHONE: +1.914.948.6200

FAX: +1.914.948.6270

You may also contact us at:

SMARTS
44 South Broadway
White Plains, New York 10601 U.S.A.

Introduction

This chapter describes the purpose of the InCharge XML Adapter (XML Adapter) and the files that make up its framework.

Purpose of the XML Adapter

An InCharge Adapter controls the communication of information between devices or applications and an InCharge Manager. An adapter normalizes topology and event information from sources other than InCharge. Typical sources include SNMP traps, or system log files. Once normalized to the ICIM data model, the information is transferred to an InCharge Manager (including either Service Assurance Global Manager, InCharge Domain Manager, or Security Infrastructure Manager).

Adapters work with third-party applications in a variety of ways to import topology and event information and prepare it for use by the Global Manager. The XML Adapter is ideal for exchanging topology information with a system for which there is no native InCharge adapter. The XML Adapter can import topology into a Global Manager or underlying domain as well as export from a Global Manager or Domain Manager. Using the XML Adapter, topology information can be exchanged between Global Managers or underlying domains.

XML Adapter Process

When importing topology into InCharge, you start with an XML document that complies with the ICIM/XML standard and represents any topological elements managed by InCharge. The XML document is imported into the InCharge Manager using the XML Adapter and the objects are migrated into the repository of the InCharge Manager. Figure 1 illustrates the XML Adapter inflow process.

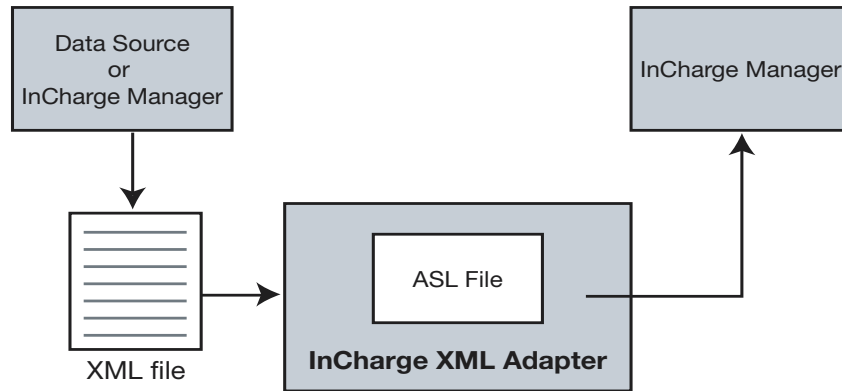


Figure 1: InCharge XML Adapter Import Process

The XML Adapter can import a large set of topology elements in one process or it can be used to make incremental updates to existing objects into the repository.

When exporting topology from InCharge, the XML Adapter extracts topological information from the InCharge Manager and produces an ICIM/XML-compliant document. Figure 2 illustrates the export process for the XML Adapter.

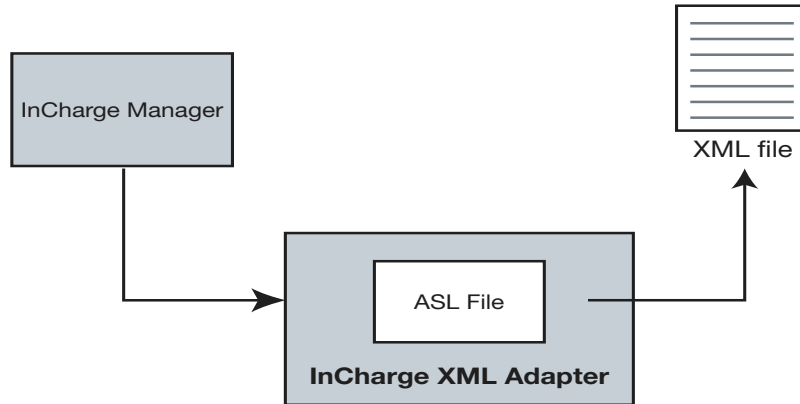


Figure 2: InCharge XML Adapter Export Process

The XML Adapter provides a structured method for transferring information in and out of a running InCharge Manager. You can use it to perform tasks such as adding or deleting objects, setting the value of attributes, and inserting objects in or removing objects from relationships.

Framework Description

- ICIM/XML DTD specification
- ICIM/XML document
- Command-line Interface (`sm_xml`)
- ASL script

ICIM/XML DTD Specification

The Document Type Definition (DTD) defines the building blocks of the XML document structure. It provides a list of allowable elements that can be used in the XML file. All of the XML tags are formally described in this document. The DTD is declared as an external document.

Attributes are used to associate name-value pairs within elements. Attributes may appear only within open tags of the element. Attribute-list declarations in the DTD are used to define the set of attributes pertaining to a given element type or to provide default values for attributes.

The type of information associated with the source objects must be identical to the type of information associated with objects in the InCharge repository. For example, if you want to import objects from an existing data source, you must make sure that the data source and the target InCharge repository database contain the same information in the same format, prior to importing. If the information models do not correspond exactly, you will lose data. The DTD specification is a guide to correctly map your data to objects to the InCharge repository.

For more detailed information on the ICIM/XML DTD, refer to [ICIM/XML DTD Specification](#) on page 14.

ICIM/XML Document

Each XML document has both a physical and a logical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity and is structured with objects, attributes and relationships. Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup.

The XML DTD contains markup declarations that provide a grammar for the XML documents.

Each XML document contains one or more elements, the boundaries of which are delimited by open tags and close tags. Each element has a type (object, attribute, or relationship) and may have a set of attribute specifications. Each attribute specification has a name and a value.

For more information on the structure of the ICIM/XML document, refer to [ICIM XML Elements](#) on page 7.

Command Line Interface (sm_xml)

The sm_xml command line utility is used to perform the import and export functions to the InCharge Manager. The file *sm_xml* (*sm_xml.exe* on Windows) is found in **BASEDIR/smarts/bin**. This interface can receive information from InCharge and send the output to a file or to standard output. It can also import a file of defined objects into the InCharge repository.

XML Export ASL Script

When exporting InCharge repository objects to XML, this adapter script extracts the InCharge objects to be represented in the XML output file. This ASL file utilizes the Adapter Scripting Language in addition to some unique commands for extracting the objects into an XML output file. A sample file called *xml-export-sample.asl* is located in **BASEDIR**/*smarts/rules/xml-if*.

For more information on exporting InCharge information to XML, refer to [Exporting InCharge Information to XML](#) on page 25.

Structure of the XML Document

The basic building blocks of an XML document are elements, attributes, and values. "Tag" is a general term that can be used for elements and attributes. A relevant sample of the ICIM/XML DTD is provided with each of the following descriptions.

Element

An element is a tag that is a container for subelements and attributes. For example, the Object element contains defining information for the instance you are adding into the repository.

The root element, <icim>, is shown as the opening and closing tags for the entire file. For example, the ICIM/XML DTD allows you to import and export *object*, *relationship*, and *attribute* elements:

```
<!ELEMENT object ( attribute | relationship )* >
```

```
<!ELEMENT relationship ( object )* >
```

```
<!ELEMENT attribute (  
    char | string | int | unsignedint | short |  
    unsignedshort | long | unsignedlong | boolean | struct  
    | float | double )* >
```

Attribute Declaration (Attribute List)

Attribute declarations, as defined in the ATTLIST of the DTD, flesh out the details of an element. All elements utilize a *name* declaration to identify it as well as a *method* declaration that instructs the adapter on whether to insert, update, or delete information about the element.

An attribute, as defined by the Attribute List in the DTD help identify the object instance. This should not be confused with an ICIM object attribute which describes the particular instance.

```
<!ATTLIST object
    class      CDATA #IMPLIED
    name       CDATA #IMPLIED
    method     ( delete | update ) "update" >

<!ATTLIST relationship
    name       CDATA #REQUIRED
    method     ( insert | remove | put ) "put" >
```

Value

The value setting of the attribute declaration is paired with the declaration in "name=value" groupings. The values define the properties of the element you are adding, modifying, or deleting in the repository.

In the following example, the Object element is made up of an element start tag (<object>), attributes (Class, Name, and Method), values "Application," "Banking," and "update"), and a close tag for the element (</Object>):

```
<object class="Application" name="Banking" method="update" />
```

Rules for a Well-Formed XML Document

The following syntax rules must be followed when creating your XML document:

- Root element contains all other elements
- Elements and subelements are properly nested:
`<A>` not `<A>`
- All elements have start and close tags:
`<object>Router</object>` as opposed to `<object>Router`
- Start and close tags should match cases:
`word` not `word`
- Attribute declaration values should be in single or double quotes:
`<attribute name="Customer 1">` not
`<attribute name=Customer 1>`

ICIM XML Elements

This chapter describes the elements used in ICIM XML and the syntax required to use them properly. A complete version of the ICIM/XML DTD can be found in [ICIM/XML DTD Specification](#) on page 14

Element Declarations

The XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or DTD.

Within the ICIM/XML document, you will define four elements:

- ICIM Root
- Objects
- Attributes
- Relationships

For each element declaration, you may have a set of attribute declarations which can include the class, name and/or method of the element.

Note: The ICIM Root element is created for you when you generate the XML document. The remaining elements must be created.

Attribute-List Declarations

For each ICIM/XML element, there is an attribute-list declaration that defines values for that element. Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type. The attribute-list also defines whether the value is required or implied. If the value is required, you must define it in the document. If the value is implied, then there is a default value for that attribute.

The syntax of the ATTLIST is defined as:

```
<!ATTLIST <element-name> <attribute-name> <attribute-type>
default-value>
```

Class

The class attribute defines the ICIM objects with common properties, common behaviors, and similar relationships. Examples of classes are Switches, Routers, Applications, Service Subscribers, and Service Offerings. Any class that is already modelled and existing in your current repository is allowable in the XML document.

Name

This attribute identifies a unique name of the element or subelement as it shall be displayed in the repository.

Method

The method attribute is used to declare the specific topology change you want to make. In the following examples, the Service Offering named GE is being updated and deleted:

```
<object class="ServiceOffering" name="GE" method="update" />
<object class="ServiceOffering" name="GE" method="delete" />
```

Table 4 identifies the methods associated with each element type.

ELEMENT	METHOD	DESCRIPTION
<object>	update	Modifies the properties of an existing object or creates the object if it does not already exist.
	delete	Removes an object from the repository

ELEMENT	METHOD	DESCRIPTION
<attribute>	insert	Adds a value to a set of attributes.
	remove	Removes an attribute from an object.
	put	Assigns a value to an attribute.
<relationship>	insert	Adds an object to a relationship.
	remove	Removes an object from a relationship.
	put	Assigns an object or set of objects to a relationship.

Table 4: Method Descriptions for ICIM XML References

Note: A "put" will associate a value, or structure of values, with an object. If a value already exists, it will be overwritten. An "insert" will add a value to be associated with an object and will not overwrite any existing values.

Object Elements

An <object> element identifies a repository object.

In the ICIM/XML DTD specification, the object element is declared as:

```
<!ELEMENT object (attribute | relationship)>
```

This declares the InCharge object element that is going to be modified. Nested in this element will be subelements that define the attribute information associated with it. The subelement of the object element can be one or more ICIM attribute or ICIM relationship.

<Object> Element Attribute List Declarations

```
<!ATTLIST object
    class      CDATA #IMPLIED
    name       CDATA #IMPLIED
    method     ( update | delete ) "update" >
```

Table 5 identifies the attribute declarations for the <object> element. If the object you specify does not already exist in the repository you must define the class of the instance to be created.

XML ATTRIBUTE	DESCRIPTION	POSSIBLE VALUE	VALUE REQUIRED/IMPLIED
class	The ICIM class of the object being declared. This can be any ICIM class name.	CDATA	Implied Default is MR_Object
name	Name of the object instance.	CDATA	Implied No default value
method	Specifies the desired topology change. You can delete an existing object or update an existing object. An update will create the object if it does not already exist in the repository.	delete update	Implied Default is "update"

Table 5: Attribute-list Declarations for the Object Element

An object will have zero or more attributes and/or relationships within one declaration.

Attribute Elements

An attribute element specifies a property that is present for a particular object element. Attributes flesh out the details of an object. For instance, DisplayName is an attribute of the element Application and sets how the instance will be identified in the Global Console. An attribute declaration includes the attribute's type and value. The value describes the state of the instance.

The attribute element must be nested within the object element.

<Attribute> Element Attribute-List Declarations

For the <attribute> element, the ATTLIST is defined as:

```
<!ELEMENT attribute ( attribute | relationship )* >
<!ATTLIST attribute
    name          CDATA #REQUIRED
    method        ( insert | remove | put ) #IMPLIED >
```

Table 6 identifies the attribute list declarations for the <attribute> element.

XML ATTRIBUTE	DESCRIPTION	POSSIBLE VALUE	VALUE REQUIRED/IMPLIED
name	Name of the object instance.	CDATA	Required
method	Specifies the desired topology change. You can create a new attribute, remove an existing attribute or update (put) an existing attribute.	insert remove put	Implied Default is "put"

Table 6: Attribute-list Declarations for the Attribute Element

For example, the following sample identifies an Application named App-BankingDataBase having a display name of Banking.

```
<object class="Application" name="APP-BankingDataBase">
  <attribute name="DisplayName">
    <string>Banking</string>
  </attribute>
</object>
```

And in this example, the UtilizationThreshold attribute is set with an integer data type:

```
<attribute name="UtilizationThreshold">
  <int>75</int>
</attribute>
```

Applying Multiple Attributes to an Object

Since an attribute element can have one or more values within one declaration, you must be able to apply values in multiples. If the target to be modified is a table, the <attribute> element may be modified using a <struct> tag to modify multiple values. A struct is a collection of different data types. It allows you to apply a set of values to an element (such as string, integer, boolean). For example, to add two properties of different types (string, and unsignedshort) to the AgentInfoSet attribute, the following syntax is used:

```
<attribute name="AgentInfoSet" method="insert">
  <struct>
    <string>public</string>
    <unsignedshort>162</unsignedshort>
  </struct>
</attribute>
```

If an object contains an attribute called `IntegerList` which contains a list of 3 integers this attribute can be set in the following way:

```
<attribute name="IntegerList">
  <struct>
    <int>4</int>
    <int>8</int>
    <int>9</int>
  </struct>
</attribute>
```

If on the other hand the attribute represents an `IntegerList2` which has an *unspecified* number of integers it can be imported in the following manner:

```
<attribute name="IntegerList2">
  <struct><int>4</int></struct>
  <struct><int>8</int></struct>
  <struct><int>9</int></struct>
</attribute>
```

Relationship Elements

A relationship maps instances of one class to instances of the same or another class. For each relationship, an inverse relationship may also exist that maps from the related class back to the originating class.

The `<relationship>` element modifies a relationship. Like the `<attribute>` element, `<relationship>` may be used to put, insert, or remove an object from a relationship.

<Relationship> Element Attribute List Declarations

```
<!ATTLIST relationship
  name      CDATA #REQUIRED
  method    ( insert | remove | put ) #IMPLIED >
```

Table 7 identifies the attribute declarations available for the `<relationship>` element.

XML ATTRIBUTE	DESCRIPTION	POSSIBLE VALUE	VALUE REQUIRED/IMPLIED
name	Name of the object instance.	CDATA	Required
method	Specifies the desired topology change. You can create a new relationship, delete an existing relationship, or update an existing relationship.	insert remove put	IMPLIED Default is "put"

Table 7: Attribute-list Declarations for the Relationship Element

For example, to add the HostedBy relationship between the application Banking and the host BankingHost, use the following syntax:

```
<object class="Application" name="Banking">
  <relationship name="HostedBy" method="put" />
  <object class="Host" name="BankingHost" method="update" />
</relationship>
</object>
```

If multiple objects are specified when the method is "insert" (or "remove") then all of them are inserted (or removed). The methods "put" and "insert" are equivalent if the relationship is a singleton. The "remove" method may be used to clear a singleton relationship:

```
<relationship name="HostedBy" method="remove" />
```

A relationship can have zero or more objects associated with it within one declaration. In other words, you may want to add a relationship even if you do not have a specific object you want to assign to it. To declare zero objects, you may put or insert the NULL object declaration:

```
<relationship name="HostedBy">
  <object/>
</relationship>
```

ICIM/XML DTD Specification

The ICIM/XML DTD file is called *icim_xml.dtd* and is located in **BASEDIR/smarts/rules/xml-if**. The contents of this file must not be modified. The *icim_xml.dtd* file contains the following:

```

<!ELEMENT icim ( object )* >
<!ATTLIST icim source CDATA #IMPLIED >

<!ELEMENT object ( attribute | relationship )* >
<!ATTLIST object
    class      CDATA #IMPLIED
    name       CDATA #IMPLIED
    method     ( delete | update ) "update" >

<!ELEMENT relationship ( object )* >
<!ATTLIST relationship
    name       CDATA #REQUIRED
    method     ( insert | remove | put ) "put" >

<!ELEMENT attribute (
    char | string | int | unsignedint | short |
    unsignedshort | long | unsignedlong | boolean | struct
    | float | double )* >

<!ATTLIST attribute
    name       CDATA #REQUIRED
    method     ( insert | remove | put ) "put" >

<!ELEMENT string      ( #PCDATA ) >
<!ELEMENT char        ( #PCDATA ) >
<!ELEMENT int          ( #PCDATA ) >
<!ELEMENT unsignedint ( #PCDATA ) >
<!ELEMENT short       ( #PCDATA ) >
<!ELEMENT unsignedshort ( #PCDATA ) >
<!ELEMENT long        ( #PCDATA ) >
<!ELEMENT unsignedlong ( #PCDATA ) >
<!ELEMENT boolean     ( #PCDATA ) >
<!ELEMENT float       ( #PCDATA ) >
<!ELEMENT double      ( #PCDATA ) >
<!ELEMENT struct      (
    char | string | int | unsignedint | short |
    unsignedshort | long | unsignedlong | boolean |
    struct | float | double )* >

```

3

Building Topology in XML

This chapter explains how to use the ICIM XML elements to build the XML document for use in exchanging topology information with an InCharge Manager. The XML document is a kind of database. You can control the following actions using XML:

- Add topology
- Update topology
- Delete topology
- Assign values to an attribute, relationship, or relationship set
- Insert and remove information in a table attribute or relationship set

Using XML For InCharge

Working with objects in XML means that you are creating, updating and removing instances of classes in the InCharge Manager. You cannot add classes to InCharge using the XML Adapter; you can only add instances and relationships to existing classes.

This section contains XML excerpts to introduce the basic XML declarations that you will use to import topology changes. This set of examples is importing Business Impact Manager topology into Service Assurance Manager. For some of the examples, a Services Map in the Global Console illustrates the results of the topology import.

Adding and Updating Topology

This section shows examples to represent new objects, relationships, and attributes in XML.

Adding Objects

Importing instances requires that you create objects in the XML file. In the XML file, we have created two ServiceSubscriber instances, called Cust1 and Cust2.

```
<icim>
  <object class="ServiceSubscriber" name="Cust1" />
  <object class="ServiceSubscriber" name="Cust2" />
</icim>
```

By default, the method for an object element is "update" so it is not necessary to specify it. If the object does not already exist then the update method automatically adds it to the repository. Figure 3 illustrates a Services Map and topology browser after importing these Service Subscribers.

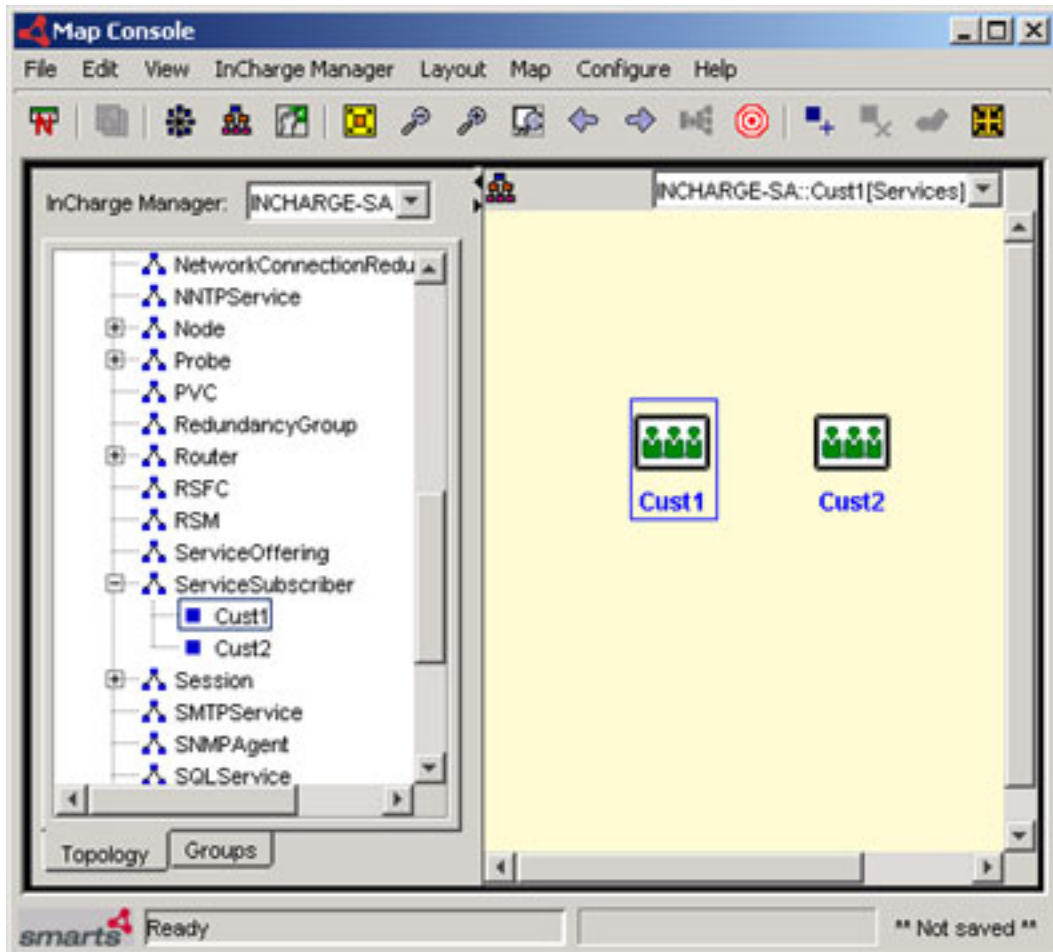


Figure 3: Map of Imported Service Subscribers

Adding Attributes

Adding attribute elements requires that you nest the attribute declaration within the appropriate object declaration. This example adds the Display Name attribute to the Service Subscribers you created. This changes the Display Name of "Cust1" to "Customer 1" and "Cust2" to "Customer 2".

```
<icim>

<object class="ServiceSubscriber" name="Cust1" method="update">
  <attribute name="DisplayName" method="put">
    <string>Customer 1</string>
  </attribute>
</object>
```

```
</object>

<object class="ServiceSubscriber" name="Cust2" method="update">
  <attribute name="DisplayName" method="put">
    <string>Customer 2</string>
  </attribute>
</object>

</icim>
```

Figure 4 illustrates the Services Subscribers with their new display names.

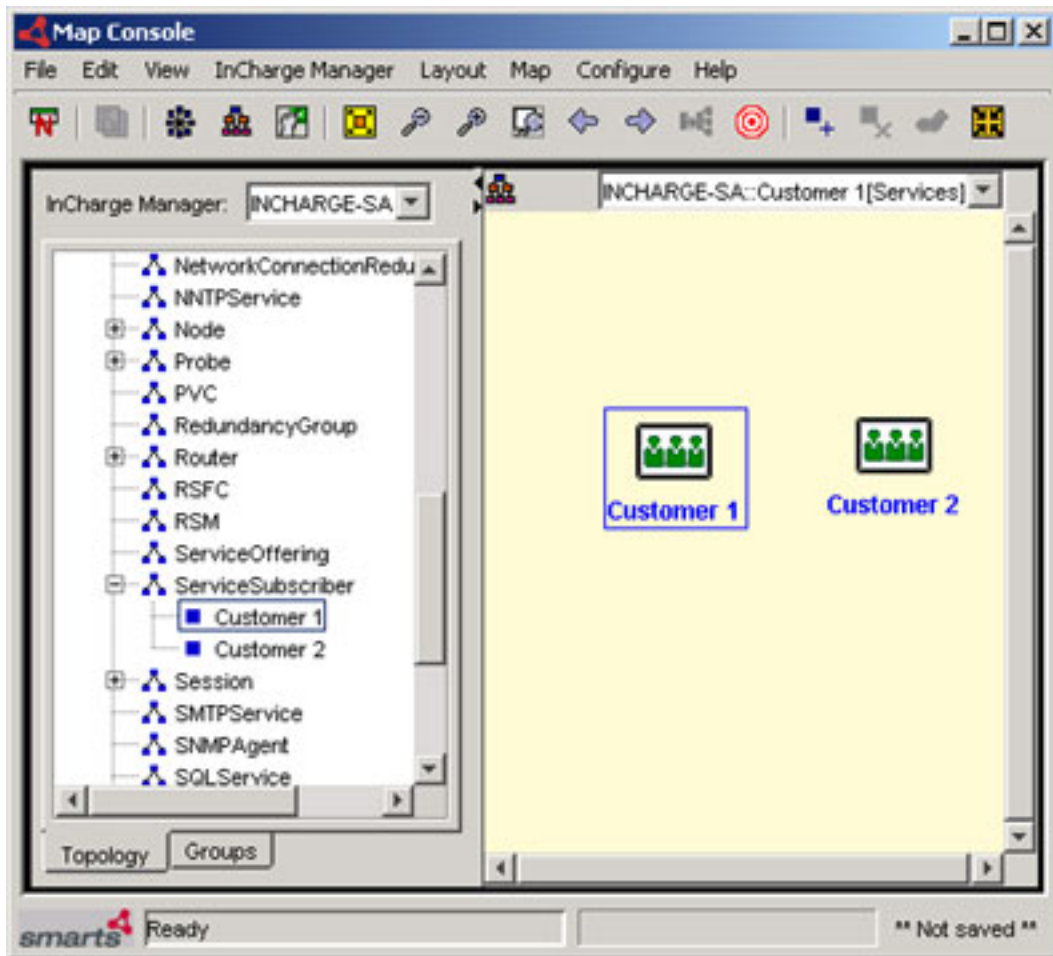


Figure 4: Map of Imported Service Subscribers with New Display Names

Adding Relationships Using the Put Method

Adding relationships requires that you nest the relationship declarations within the object declarations. Using the "put" method means you are assigning the only attribute intended for that object. The following example shows how to add a Service Offering called "EastCoast" and use the "Subscriptions" relationship to associate that offering with the Customer 1 subscriber you created previously.

```
<icim>

  <object class="ServiceSubscriber" name="Cust 1" method="update">

    <relationship name="Subscriptions" method="put">
      <object class="ServiceOffering" name="EastCoast" />
    </relationship>

  </object>

</icim>
```

Figure 5 illustrates the new Service Offering, EastCoast and Services Subscriber, Customer 1 with a Subscriptions relationship.

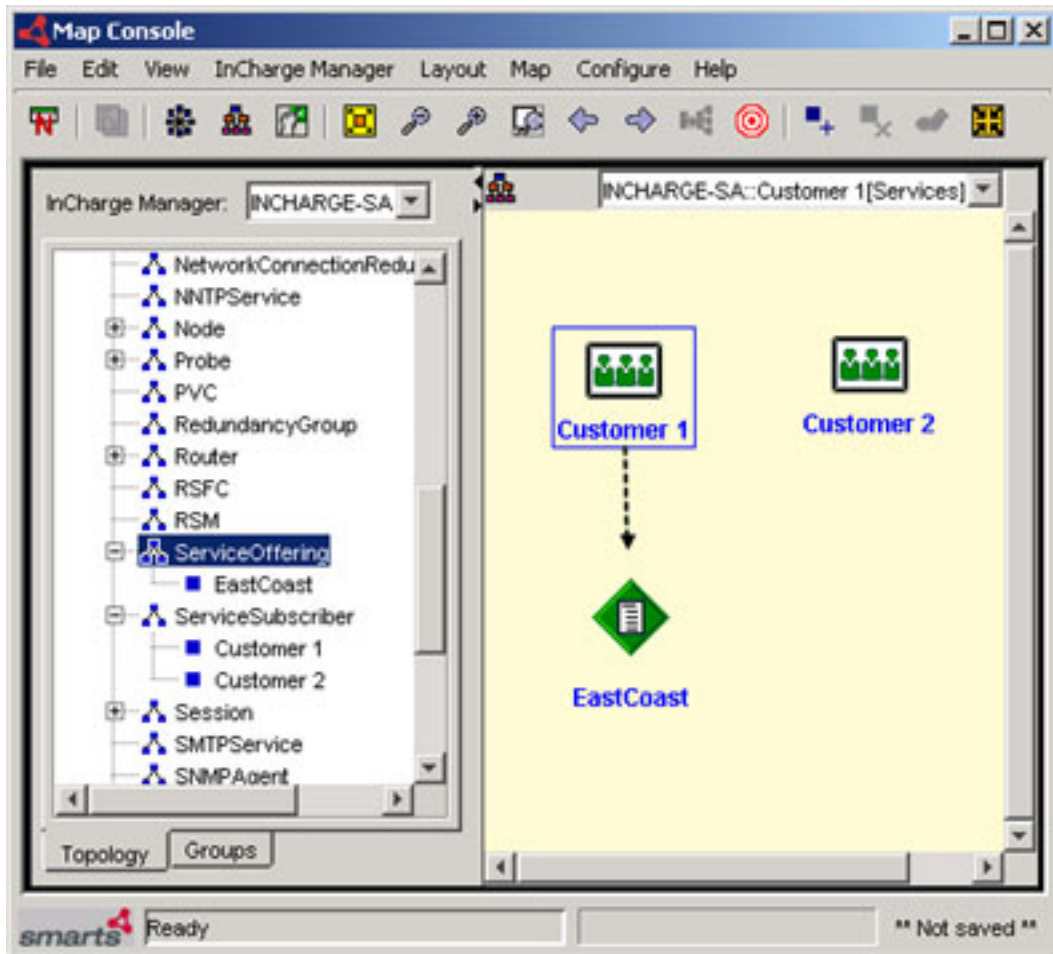


Figure 5: Map of Imported Service Subscribers and Service Offering

Adding Relationships Using the Insert Method

Using the "insert" method allows you to add to the list of attributes intended for that object. The following example shows how to insert another Service Offering called MidWest into the Subscriptions relationship for Customer 1.

```
<icim>
<object class="ServiceSubscriber" name="Cust 1" method="update">

  <relationship name="Subscriptions" method="insert">
    <object class="ServiceOffering" name="MidWest" />
  </relationship>
</object>
</icim>
```

```

</object>
</icim>

```

Figure 6 illustrates the new Service Offering, MidWest, and Services Subscriber, Customer 1 with a Subscriptions relationship.

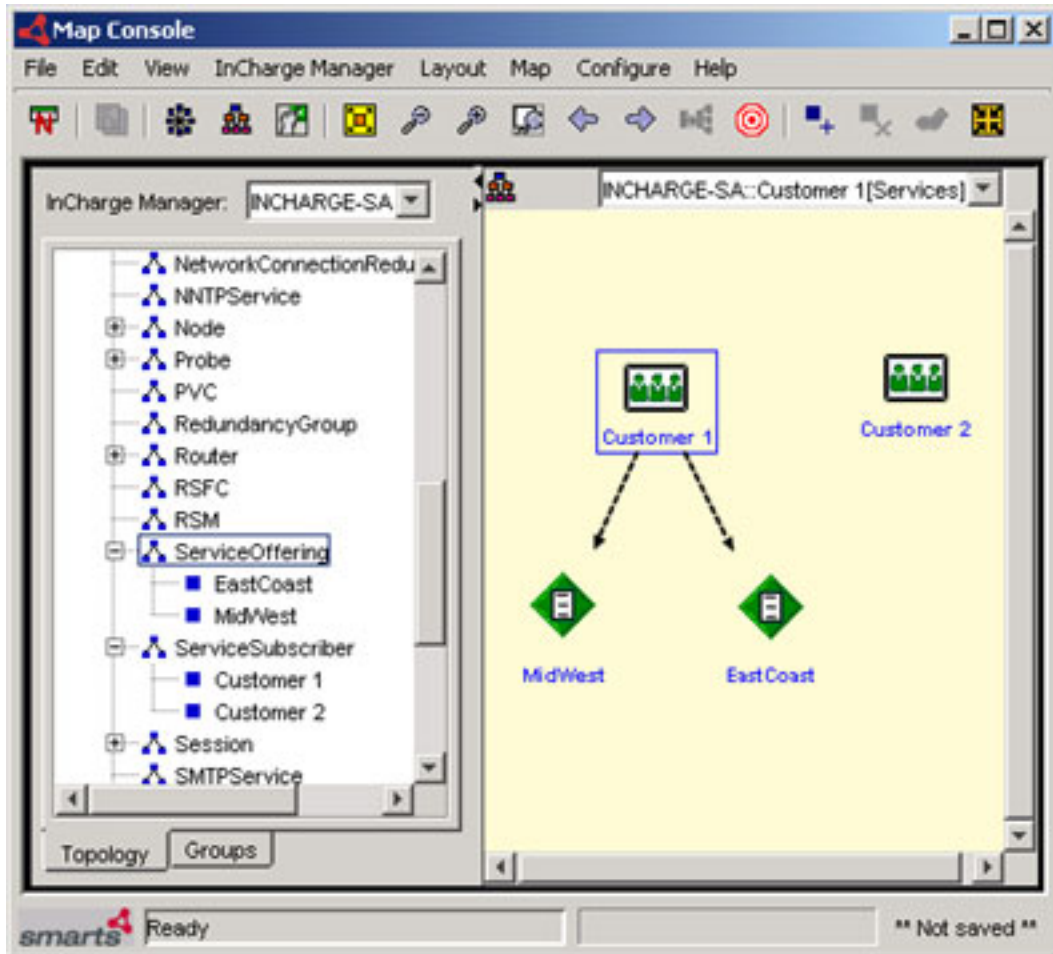


Figure 6: Map of Imported Service Subscribers and Service Offering

Removing Topology

This section describes how to remove topology from an InCharge Manager.

Attempting to perform the following tasks will cause a runtime error when you import the XML file:

- Deleting a non-existent object
- Deleting a non-existent attribute or relationship

Removing Objects

To remove an object from the repository, use the "delete" method for an object element that already exists. For example, to delete the Service Subscriber called Cust 2, use the following syntax:

```
<object class="ServiceSubscriber" name="Cust 2" method="delete" />
```

Removing Attributes and Relationships

To remove a particular attribute or relationship for an object, adhere to the syntax of the ICIM/XML DTD. This means that the relationship and attribute updates must be nested within the appropriate object declaration. For example, to remove the Subscriptions relationship from the EastCoast service offering, use the following syntax:

```
<icim>  
  
  <object class="ServiceSubscriber" name="Cust 1" method="update">  
    <relationship name="Subscriptions" method="remove">  
      <object class="ServiceOffering" name="EastCoast" />  
    </relationship>  
  </object>
```

The topology returns to look as it did in Figure 5.

Data Types

Table 8 lists the data types supported in the ICIM/XML DTD.

VALUE	DESCRIPTION
boolean	Datatype that is defined as two possible values (such as 0 or 1, on or off, true or false)
char	Any single ASCII character
double	Any double-precision floating point number (similar to C++)
float	Any single-precision floating point number (similar to C++)
int	A 32-bit integer ranging from -2^{31} to $(2^{31} - 1)$
long	A 64-bit integer ranging from -2^{63} to $(2^{63} - 1)$
short	A 16-bit integer ranging from -32,768 to 32,767
string	An ordered list of characters
struct	A collection of fields, each field specifies its own type
unsignedint	Unsigned 32-bit integer $2^{32}-1$
unsignedlong	Unsigned 64-bit integer ranging from $2^{64}-1$ to $+2^{63} - 1$
unsignedshort	Unsigned 16-bit integer $2^{16}-1$

Table 8: Supported Data Types

4

Exporting InCharge Information to XML

Exporting object data to an XML file is a convenient way to move and store InCharge repository information in a format that can be used across the web or by another InCharge Manager.

To export data to XML you must define an ASL script that queries the InCharge repository. In the ASL script, you specify the information and matching criteria for the objects to be gathered and exported. There is a sample ASL called *xml-export-sample.asl* provided in **BASEDIR**/*smarts/rules/xml-if*.

The XML Adapter invokes the specified ASL script against the specified InCharge Manager and produces the specified XML output file.

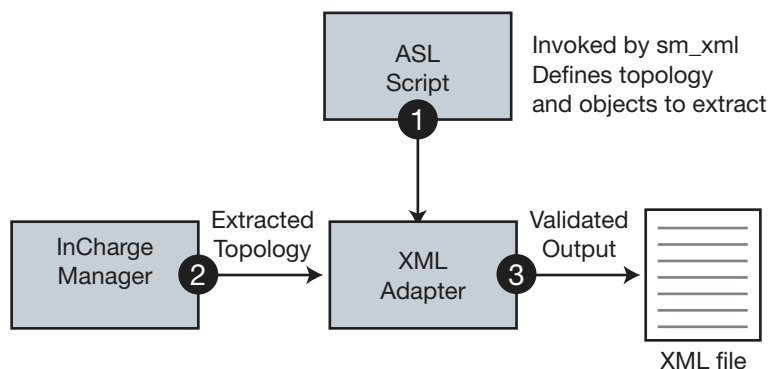


Figure 7: Process Diagram: Exporting InCharge Data in XML

The ASL Export Script

A template for the ASL export script, called *xml-export-sample.asl*, is installed in **BASEDIR**/*smarts/rules/xml-if*. Use the Adapter Scripting Language (ASL) to modify the functionality of the local copy of the script. Use the `sm_edit` utility to modify the ASL script so that a copy is made in the local directory and the proper permissions are associated to it. To open a file called *xml-export-sample.asl* using `sm_edit`:

```
▼#BASEDIR/smarts/bin>sm_edit BASEDIR/smarts/rules/xml-if/xml-export-sample.asl ▲
```

-
- ▼▲ This symbol indicates that the command should be typed on one line.
-

Save this file with a new name, replacing "sample" with an appropriate grouping name (for example, "Services" for an ASL file intended to gather information on Service Subscribers and Service Offerings).

Understanding the ASL Script

Note: You should have a strong understanding of ASL to work in the export script. For more information on working with the Adapter Scripting Language, refer to *Building Adapters with ASL*.

The first rule executed in the ASL script is the START rule. The START rule runs repeatedly until all of the data input is processed. This rule invokes other rules or can specify which objects, attributes, and relationships to export.

Data input gets processed as it is matched with all of the patterns of a rule. The START rule has components in which the request details are defined.

The XML headers and the ICIM root element are added automatically to the XML document by the `sm_xml` adapter. The ASL script is responsible for gathering and adding the object, attribute and relationship elements to the XML file.

Any modifications to the action blocks are made after the "NOTE: Customize the script from this point on" comment. These modifications shall define the specific information you want retrieved from the InCharge repository.

After the ASL script is executed, the closing tag for the ICIM root element is added to the document automatically.

Building the ASL Export Script

This section describes how to retrieve specific instances of objects, build `foreach` statements and use the builder statements to export the exact information you need from the InCharge repository.

The ASL file is constructed from these functions and statements: `getInstance`, `foreach` statements and builder statements.

getInstance Function

The `getInstances` function returns a list of object names for the given class. The syntax of the `getInstances` function is:

```
listname = getInstances(<classname>);
```

For example, in order to retrieve all of the instances of the `ServiceSubscriber` class and add them to your XML document, you would begin with the following line:

```
EXPORT_SERVICESUBSCRIBER {
} do {
    svcSubList = getInstances("ServiceSubscriber");
    ...
}
```

Once you have defined which objects you want to export, you must define how to handle them. This is done using the builder statements described in [Builder Statements](#) on page 28 and the `foreach` statements described below.

Foreach Statement

The `foreach` statement iterates through all of the members of a list or a table. The syntax of the `foreach` statement is:

```
foreach variable (listortablename) {statements}
```

When the `foreach` statement is used with a loop, the variable stores the value of each member of the list every time the `foreach` statement loops. The members returned are in numerical order by their list indices.

Building on our `ServiceSubscriber` example above, add the following statements:

```
EXPORT_SERVICESUBSCRIBER {
} do {
    svcSubList = getInstances("ServiceSubscriber")? LOG, NEXT;
    foreach svcSub (svcSubList) {
        builder->addObject(svcSub, "ServiceSubscriber");
    }
}
```

```

        builder->addAttribute(svcSub, "DisplayName");
        builder->addObjectClosing();
    }
}

```

Builder Statements

Each element is created and formatted in the ICIM/XML document using builder statements that define the objects, attributes and relationships. The syntax for the builder statements is as follows:

```
builder-> <BuilderStatement>( <Argument>, "<Value>");
```

Object Element Type Builder Statements

An object element requires two builder statements. Table 9 describes the builder statements and arguments for an object element.

BUILDER STATEMENT	DESCRIPTION	ARGUMENTS	REQUIRED / OPTIONAL
addObject()	Adds the instance to the XML document. It creates the open tag for the object declaration.	Instance Name	Optional
		Class Name	Required
		Method	Optional
addObjectClosing()	Adds the closing tag for the object declaration in the XML file.	No Arguments	N/A
addNullObject	Adds a null object "<object/>" into the output XML document.	No Argument	N/A

Table 9: Builder Statements for the Object Element Type

For example, to export an instance of the ServiceSubscriber class to the XML document, the builder statement would look something like this:

```

builder->addObject(svcSub, "ServiceSubscriber");
builder->addObjectClosing();
name="\DisplayName\>");
    wrapper->addToDocument("<value>");
    wrapper->addToDocument("<string>".wrapper-
>normalizeForXML(svcSubObj->DisplayName). "</string>");
    wrapper->addToDocument("</value>");

```

Relationship Element Type Builder Statements

A relationship element has two required builder statements. Table 10 describes the builder statements and arguments for a relationship element.

BUILDER STATEMENT	DESCRIPTION	ARGUMENTS	REQUIRED / OPTIONAL
addRelationship()	This adds the relationship to the XML document. It creates the open tag for the relationship declaration.	Relationship Name	Required
		Method	Optional
addRelationshipClosing ()	Adds the closing tag for the relationship declaration in the XML file.	No Arguments	N/A

Table 10: Builder Statements for the Relationship Element Type

For example, to export all of the objects with a Subscriptions relationship to the XML document, the builder statement would look something like this:

```
builder->addRelationship("Subscriptions");
    foreach...
builder->addRelationshipClosing();
```

Attribute Element Type Builder Statements

An attribute element has only one required builder statement. Table 11 describes the builder statement and arguments for an attribute element.

BUILDER STATEMENT	DESCRIPTION	ARGUMENTS	REQUIRED / OPTIONAL
addAttribute()	This adds the attributes of the instance to the XML document.	Instance Name	Required
		Attribute Name	Required
		Method	Optional

Table 11: Builder Statements for the Attribute Element Type

Note that the closing builder statement is not needed for this element because there is no recursive part so we can predict the end of the statement.

Since the ICIM/XML DTD requires that attribute definitions are nested within an object definition, this builder statement is defined with an object element. For example, to export instances of the ServiceSubscriber class, with their display names, to the XML document the builder statement would look something like this:

```
builder->addObject(svcSub, "ServiceSubscriber");
builder->addAttribute(svcSub, "DisplayName");
builder->addObjectClosing();
```

String Builder Statement

Optionally, you can add text into the XML document along with any of the declarations. After any of the builder statements defined above, you can use the addText statement. The argument may be any string that complies with the ICIM/XML DTD. Table 11 describes the builder statement and arguments for this element.

BUILDER STATEMENT	DESCRIPTION	ARGUMENTS	REQUIRED / OPTIONAL
addText ()	Adds optional text to the XML declaration.	String	Required

Table 12: Builder Statements for the String.

For example, to export instances of the ServiceSubscriber class, with their display names, and additional text that says Sample Offering to the XML document, use the following syntax:

```
builder->addObject (svcSub, "ServiceSubscriber");
builder->addAttribute (svcSub, "DisplayName");
builder->addObjectClosing ();
builder->addText("<object class=\"ServiceOffering\" \
name=\"Sample Offering\" method=\"update\" />");
```

Parsing the XML

The resulting XML generated by the ASL script and the XML Adapter is passed through a validating parser. This is to ensure compliance with the ICIM/XML DTD. This parser, enables the XML Adapter to read and write XML data.

SMARTS provides the Xerces XML Parser with the installation of the XML Adapter.

Running sm_xml for Export

There are four export-specific parameters associated with the sm_xml utility that you may need to use

PARAMETER	DESCRIPTION	OPTIONAL FLAG
OPTIONS:		
<code>--server=<name></code>	Name of the InCharge Manager from which you are retrieving repository information. By default, this InCharge Manager name is INCHARGE.	<code>-s</code>
<code>--broker=<location></code>	Alternate location of the InCharge broker. Must be typed as <code>host:port</code> . The default is the broker specified for the InCharge Manager at installation.	<code>-b</code>
COMMANDS:		
<code>export --xmlfile=<output-xmlfile> <asl-script></code>	Export an XML document using the XML export ASL script named <code><asl-script></code> . By default, the XML is written to standard output. Optionally, the XML can be written to a file if you specify <code>--xmlfile=<output-xmlfile></code> .	<code>-x <output-xmlfile></code>
COMMAND ARGUMENTS:		
<code>output=<xmlfile></code>	Name of the resulting XML file (and qualified path to save the file). Optionally, you can choose not to define a directory and file name and the resulting XML will be displayed in standard output.	<code>-x</code>
<code><asl-script></code>	The ASL rule script used to export the InCharge topology. The adapter will look for the ASL script in BASEDIR / <code>local/rules/xml-if</code> first. If it is not found there, it will locate it in BASEDIR / <code>rules/xml-if</code>	N/A

Table 13: XML Exporter Command Parameters

The resulting XML is passed through a validating parser. If the document is non-compliant with the ICIM/XML DTD an error will display and the program will exit with a non-zero exit status. When an error occurs, the XML is still written to standard output.

For example, to generate an XML file called *InChargeApplications.xml* containing information from an InCharge Manager called INCHARGE, with an ASL export script called *xml-export-Applications.asl*, use the following command:

```
▼sm_xml --server=<INCHARGE> export --
xmlfile=InChargeApplications.xml xml-export-
Applications.asl▲
```

▼▲ This symbol indicates that the command should be typed all on one line.

Sample ASL Export Script

In this example ASL script, you are requesting XML output for all instances of the ServiceSubscriber class, the ServiceOfferings to which they are subscribed, and the Applications that compose the ServiceOfferings.

```
/* xml-export-sample.asl
*
* Copyright (C) 2003, System Management ARTS (SMARTS)
* All Rights Reserved
*
* Sample Rule Set to export the topology of the ServiceSubscribers, the
* ServiceOfferings they are subscribed to, the Applications that these
* ServiceOfferings are composedOf, and the Hosts that Host these
* Applications from the server into an XML document that will conform
* to the ICIM/XML DTD specifications.
*
* This is a sample xml-export-*.asl file. Users should make copies of this
* file, and edit them according to their needs. But they should always name
* these asl files according to the Naming convention, xml-export-<name>.asl
*/

/*
* the XML_Builder object
*/
default builderName = "";
aslname = " ".getRuleFileName().": ";
builder = self->object("XML_Builder", builderName)? LOG, STOP;
if (builder->isNull()) {
    print(aslname."ERROR: The Wrapper object has not been created.");
    print(aslname."ERROR: Please make sure the CREATE_BUILDER is called ".
        "before calling the INITIALIZE rule in the script.");
    stop();
}

/*
* NOTE: Customize the script from this point on.
*/

START() {
    .. eol
    EXPORT_BIM_INSTANCES
}

EXPORT_BIM_INSTANCES {
    EXPORT_SERVICESUBSCRIBER
    EXPORT_SERVICEOFFERRING
    EXPORT_APPLICATION
}
```

```

EXPORT_SERVICESUBSCRIBER {
} do {
  /*
  * Get all the instances of ServiceSubscriber class and start adding to the
  * XML Document.
  */
  svcSubList = getInstances("ServiceSubscriber")? LOG, NEXT;
  foreach svcSub (svcSubList) {
    svcSubObj = object("ServiceSubscriber", svcSub)? LOG, IGNORE;
    builder->addObject(svcSub, "ServiceSubscriber");
    builder->addAttribute(svcSub, "DisplayName");
    builder->addRelationship("Subscriptions");
    /*
    * Get all the ServiceOffering objects that each of these
    * ServiceSubscribers subscribe to, and add them to the document.
    */
    foreach svcOffObj (svcSubObj->Subscriptions) {
      builder->addObject(svcOffObj->Name, svcOffObj->CreationClassName);
      builder->addObjectClosing();
    }
    builder->addRelationshipClosing();
    builder->addObjectClosing();
  }
}

EXPORT_SERVICEOFFERING {
} do {
  /*
  * Get all the instances of ServiceOffering class and start adding to the
  * XML Document.
  */
  svcOffList = getInstances("ServiceOffering")? LOG, NEXT;
  foreach svcOff (svcOffList) {
    svcOffObj = object("ServiceOffering", svcOff)? LOG, IGNORE;
    builder->addObject(svcOff, "ServiceOffering");
    builder->addAttribute(svcOff, "DisplayName");
    builder->addRelationship("ConsistsOf");
    /*
    * Get all the <infrastructure> objects that each of these
    * ServiceOfferings consists of, and add them to the document.
    */
    foreach infObj (svcOffObj->ConsistsOf) {
      builder->addObject(infObj->Name, infObj->CreationClassName);
      builder->addObjectClosing();
    }
    builder->addRelationshipClosing();
    builder->addObjectClosing();
  }
}

```

```
EXPORT_APPLICATION {
} do {
  /*
  * Get all the instances of Application class and start adding to the
  * XML Document.
  */
  appList = getInstances("Application"? LOG, NEXT;
  foreach app (appList) {
    appObj = object("Application", app)? LOG, IGNORE;
    builder->addObject(app, "Application");
    builder->addAttribute(app, "DisplayName");

    /*
    * Get the Host object that this Application is HostedBy,
    * and add it to the document.
    */
    hostObj = appObj->HostedBy;
    builder->addRelationship("HostedBy");
    builder->addObject(hostObj->Name, hostObj->CreationClassName);
    builder->addObjectClosing();
    builder->addRelationshipClosing();

    /*
    * Get all the Transaction objects that these Applications Produce and
    * Consume, and export them to the XML Document also.
    */
    builder->addRelationship("Produces");
    foreach trans (appObj->Produces) {
      builder->addObjectClosing();
    }
    builder->addRelationshipClosing();

    builder->addRelationship("Consumes");
    foreach trans (appObj->Consumes) {
      builder->addObject(trans->Name, trans->CreationClassName);
      builder->addObjectClosing();
    }
    builder->addRelationshipClosing();

    builder->addObjectClosing();
  }
}
```

5

Importing Information Into InCharge

This chapter describes how to import XML data into the InCharge repository. The following details are discussed:

- A description of the import process
- How to verify XML data before importing
- How to import the XML data

XML Import Process

To import an XML document containing topology, events or other objects, your XML document must comply with the ICIM/XML DTD. Figure 8 illustrates the XML import process. You may have a utility that exports the desired topology data as XML using a unique DTD as in Option A. If this is the case, you will have to use an XSL Transformation (XSLT) which is a language for transforming XML documents into other XML documents.

Alternatively, you can manually create an XML file that complies with the ICIM/XML DTD as in Option B.

When your XML document is ready, you will use the `sm_xml` adapter to import the new objects into the InCharge repository. Figure 8 displays the import process:

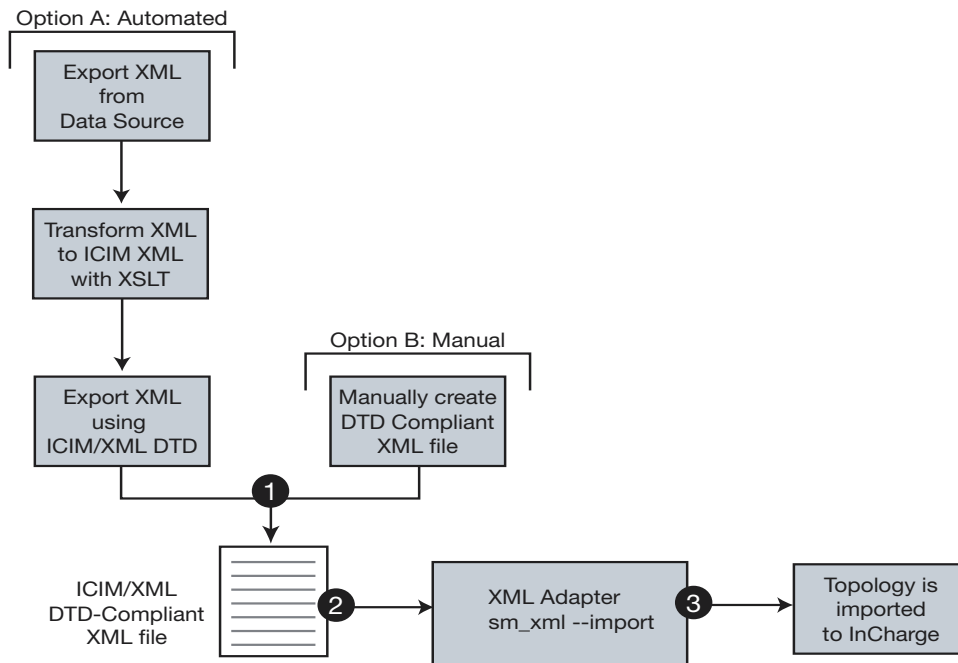


Figure 8: Process Diagram: Importing XML Data into InCharge

Before Importing XML

You do not need to shut down the InCharge Manager before you import the XML file. However, when importing, you should verify that your XML is valid according to the ICIM/XML DTD. The DTD file, called *icim_xml.dtd* is located in **BASEDIR**/*smarts/conf/xml-if*.

A runtime error is generated if the XML importer cannot execute the instructions encoded in the XML document. By default, the importer skips any element (or child element) containing an error. It continues with the next element and generates a report of errors at the end. To determine if the XML is valid and complies with the ICIM/XML DTD, run the following command:

```
▼ sm_xml --server=<INCHARGE> --import=<Applications.xml>
--verify ▲
```

▼▲ This symbol indicates that the command should be typed on one line.

If there are any errors within your XML document, they are listed so you can address them.

Note: If you create an object with read-only attributes (such as `ICS_Notification`), those attributes may cause a runtime error.

Running `sm_xml` to Import XML

Table 14 describes the command parameters you can use with `sm_xml`:

PARAMETER	DESCRIPTION	OPTIONAL FLAG
OPTIONS:		
<code>--server=<name></code>	Name of the target InCharge Manager. By default, this Manager name is INCHARGE.	<code>-s</code>
<code>--broker=<location></code>	Alternate location of the InCharge broker. Must be typed as <code>host:port</code> . The default is the broker specified for the InCharge Manager at installation.	<code>-b</code>
COMMANDS:		
<code>import [--exit-on-error] [--verify] <input-xmlfile></code>	Import an XML document called <code><input-xmlfile></code> . Optionally, the XML can be verified without importing it to the Manager using <code>--verify</code> . Additionally, using <code>--exit-on-error</code> , the XML can be imported but the import process will stop when an error is detected.	
COMMAND ARGUMENTS:		
<code><input-xmlfile></code>	The XML file that is used to import the InCharge topology.	N/A
<code>--exit-on-error</code>	This option allows you to import the XML but stops the import process when a runtime error is encountered. By default, without this command option, a runtime error is reported but the import process resumes at the next element. Note that anything imported prior to the runtime error remains in InCharge repository.	<code>-e</code>
<code>--verify</code>	This option validates the XML file against the ICIM/XML DTD specification but does not import the XML to the InCharge Manager.	N/A

Table 14: XML Exporter Command Parameters

For example, to import a file called *Applications.xml* into an InCharge Manager called INCHARGE, use the following command:

```
sm_xml import --exit-on-error Applications.xml
```

Handling Runtime Errors

A runtime error is generated if the XML importer cannot execute the instructions encoded in the XML document. By default, the importer skips any element (or subelement) containing an error. It continues with the next element and generates a report of errors at the end.

There are two ways to avoid the runtime error:

- `--verify`: Without importing the ICIM/XML document, you can run the XML you produced to see if it will be validated by the parser.
- `--exit-on-error`: By default, objects will be imported as they are interpreted from the XML document and any objects that are invalid will be identified in a summary file. Use this command option to quit the import process when the adapter detects an error. Note that any objects imported before the error was detected will remain in the repository.

Index

A

- Adapter
 - Definition 1
 - Process 2
 - Purpose 1
 - sm_edit 26
 - sm_xml 26
 - Export 30
 - Import 35
 - Running 37
 - XML 1
- ASL
 - Export script 26
 - getinstance function 27
 - Start rule 26
 - XML export script 5
- Attribute
 - Declaration 5
 - Declarations 8
 - Class 8
 - Method 8
 - Name 8
 - Definition 10
 - Removing 22

B

- BASEDIR xi
- Builder statements 28
 - Attribute element type 29
 - Object element type 28
 - Relationship element type 28
 - String builder 29, 30

D

- Data types
 - ICIM/XML 23
- Declarations
 - Attribute 8
 - Class 8
 - Method 8
 - Name 8
- DTD

- ICIM/XML
 - Specification 14

E

- Element
 - Attribute 10
 - Applying multiple 11
 - Declarations 7
 - Definition 5
 - Object 9
 - Relationship 12
- End tag 6
- Export Script
 - Building
 - getinstance function 27
 - Sample 32
 - Understanding 26
 - Export script 26
 - Building 27
 - Builder statements 28
 - Foreach statement 27
- Exporting XML
 - Process 2

F

- Foreach statement 27

G

- getinstance function 27

I

- ICIM/XML
 - Data types 23
 - Document description 4
 - DTD 3
 - Specification 14
 - Overview 5
 - Root element 6
 - Standard 2
- Import
 - XML 35

Importing
 Process 2
Insert method 20

P

Parsing XML 30
 Xerces 30
Put method 19

R

Removing attributes 22
Removing relationships 22
Removing topology 21

S

Script
 ASL export 26
sm_xml 4
 Export 30
 Import 35
 Interface 4
 Running 37
Start tag 6

T

Technical support xiv
Topology
 Adding 16
 Adding attributes 17
 Adding objects 16
 Adding relationships
 Insert method 20
 Put method 19
 Remove 21
 Removing objects 22
 Updating 16

V

Value declaration 6

X

XML
 Actions 15
 Adapter 1
 Composition 4
 End tag 6
 Exporting process description 2

Guidelines for writing 6
Import
 Runtime errors 38
Import process 35
Importing process description 2
Logical structure 4
Nesting elements 6
Parsing 30
 Xerces 30
Physical structure 4
Root element 6
Start tag 6
Using 16
Validation 36
XSL Transformation (XSLT) 35