



EMC[®] Watch4net[®]

Version 6.3u2

Smarts Collector Topology Importer Guide

P/N: pending

Rev : 40032

EMC²

Copyright ©2013 - 2014 EMC Corporation. All rights reserved. Published in the USA.

Published April, 2014

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

EMC², EMC, and the EMC logo are registered trademarks or trademarks of EMC Corporation in the United States and other countries. All other trademarks used herein are the property of their respective owners.

For the most up-to-date regulatory document for your product line, go to EMC Online Support (<https://support.emc.com>).

Contents

Preface	iii
Document Organization	iii
Where to get help	v
1 General configuration	1
1 Topology main configuration file (topology.xml)	1
2 Topology database configuration file (topology-db.xml)	2
3 Topology domain configuration (Example topology-network.xml)	3
4 Topology fetching configuration file (Example network.xml)	3
2 Examples	7
1 Example1 : Direct relation	7
2 Example2 : NetworkConnection fork branching	8
3 Topology properties	11

Preface

Document Organization

1. General configuration	This section describes the <i>EMC² Smarts InCharge</i> smarts topology importer configuration.
2. Examples	This section provides examples for the user to refer to
3. Topology properties	This section explains the necessary internal properties to create topology links

Where to get help

EMC support, product, and licensing information can be obtained as follows:

PRODUCT INFORMATION - For documentation, release notes, software updates, or information about EMC products, go to EMC Online Support at:

<https://support.emc.com>

TECHNICAL SUPPORT - Go to EMC Online Support and click Service Center. You will see several options for contacting EMC Technical Support. Note that to open a service request, you must have a valid support agreement. Contact your EMC sales representative for details about obtaining a valid support agreement or with questions about your account.

1

General configuration

The *EMC² Smarts InCharge* topology importer retrieves data from any domain to add topology informations to the suite. It is highly flexible and customizable to fit any situation.

To use this importer, simply call the script in the bin folder of the Smarts collector module with the topology configuration as sole parameter.

```
/opt/APG/Collecting/Smarts-Collector/Default/bin/smarts-topo-importer.sh conf/topology-db.xml  
conf/topology.xml
```

Note that the base path is the module instance root folder, in this case `/opt/APG/Collecting/Smarts-Collector/Default`. All the relative references to configurations in other configuration files are relative to the module instance root folder as well.

1 Topology main configuration file (`topology.xml`)

This configuration is the main configuration file. It has the exact same format and mechanism as the discovery configuration from the Smarts Collector.

```

<config>
  <!-- ... -->
  <templates>
    <template>
      <!-- AM/PM Domains -->
      <features>
        <feature>Feature-DEVSTAT</feature>
        <feature>Feature-PERFORMANCE</feature>
        <feature>ICIM_Manager::ICIM-Manager::AMPM_Mode=
        AVAILABILITY_AND_PERFORMANCE_MANAGER</feature>
      </features>
      <configuration>conf/topology-network.xml</configuration>
    </template>
  </templates>
  <!-- ... -->
</config>

```

For further explanation of the configuration, please refer to the Smarts-Collector discovery configuration.

2 Topology database configuration file (topology-db.xml)

This configuration defines the database connection where the topology elements will be stored. It is used as one of the principal arguments of the importing cli tool.

```

<database-config xmlns="http://www.watch4net.com/APG/Smarts/Topology/Database"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.watch4net.com/APG/Smarts/Topology/Database_topology-database-
  config.xsd">

  <!-- Can have the same attributes as in the apg.xml in the frontend for the topology
  connection -->
  <database driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost:53306/topology?
  autoReconnect=true" username="apg"
    password="watch4net" poolPreparedStatements="true" testOnBorrow="false" testOnReturn="false"
    testWhileIdle="true"
    timeBetweenEvictionRunsMillis="10000" validationQuery="SELECT 1" validationQueryTimeout
    ="5"
    connectionProperties="cachePrepStmts=true;useServerPrepStmts=false;maintainTimeStats=false;
    dontTrackOpenResources=true;enableQueryTimeouts=false;processEscapeCodesForPrepStmts=false;
    jdbcCompliantTruncation=false;rewriteBatchedStatements=true;" />

```

```
</database-config>
```

Following is the listing of all mandatory elements and an explanation of them.

- `driverClassName` : Name of the SQL driver used to communicate with the database.
- `url` : URL defining the connection to the database.
- `username` : User name used to connect to the database.
- `password` : Password used to connect to the database. This password can be encrypted by the *APG* password encryption tool.

All the other settings are the same as the tomcat database connection settings.

3 Topology domain configuration (Example `topology-network.xml`)

This configuration file is the exact same file as the `smarts.xml` file. The only difference is the configuration file is a topology fetching configuration file described in the next sub section. When this configuration is parsed, the polling options are ignored (group, period, properties, smooth-factor)

4 Topology fetching configuration file (Example `network.xml`)

The *EMC² Smarts InCharge* topology importer also assumes that the topology groups, types and relations are already created in the topology database. These objects can be generated from the topology CLI tool.

The *EMC² Smarts InCharge* topology importer uses the same internal engine as the Smarts Collector for the polling of properties. As such, the configuration is based on the current `smarts.dtd` configuration file but allow some more options.

When the internal topology handler receives the internal engine object, the handling goes like this :

- Retrieve the group in the database, if it doesn't exist the internal engine object is rejected.
- Retrieve the source and destination types, if any of them doesn't exist, the internal engine object is rejected.
- Retrieve the relation between the source and destination type. If it doesn't exist, the internal engine object is rejected.
- Create or update the source and destination instance according to their respective expansion properties defined from it's type. The properties must be defined in the internal engine object. The property name form is as follows : `_[source—destination]Expansion{Property}`

- Create the topology link between the source and destination instance. Create the reverse link if specified.

Additional supported elements and options :

topology elements triggers the generation of a topology link.

```
<!-- SNMPAgent HostedBy/HostsServices Router/Switch/Host/FileServer (reverse) -->
<data class="SNMPAgent" containment="HostedBy" start="true" id="generated-id">
  <property name="DisplayName.Name" drop-empty="true" rename="sourcePart.sourcePart"/>
</data>

<data class="Router.Switch.Host.FileServer" from="generated-id">
  <property name="DisplayName.Name" drop-empty="true" rename="sourceDevice.sourceDevice" />
  <property name="DisplayName.Name" drop-empty="true" rename="destinationDevice.destinationDevice" />
  <topology group="Core.Networking" sourceType="SNMPAgent" destinationType="
DisplayClassName" relation="HostedBy" reverse="true"/>
</data>
```

Following is the explanation of the topology element attributes. All the attributes are either a property name to extract from the current data class or a quoted string.

- **group** : Topology group in which to add the topology link.
- **sourceType(optional)** : Name of the source instance type. If you leave it out, you must define the ".sourceType" magic property. See magic properties.
- **destinationType** : Name of the destination instance type.
- **relation** : Relation name between the source and destination type.
- **reverse(optional)** : Specifies the handler need to create the reverse relation between source and destination, according that it is defined in the topology database. Default is false.

data elements can be nested. That means every nested data will get an additional "from" constraint specifying the id of the parent data node.

```
<!-- SNMPAgent HostedBy/HostsServices Router/Switch/Host/FileServer (reverse) -->
<data class="SNMPAgent" containment="HostedBy" start="true">
  <property name="DisplayName.Name" drop-empty="true" rename="sourcePart.sourcePart"/>
  <property name="DisplayClassName" rename=".sourceType" />

  <data class="Router.Switch.Host.FileServer">
    <property name="DisplayName.Name" drop-empty="true" rename="sourceExpansionDevice.sourceExpansionDevice" />
```

```

        <property name="DisplayName.Name" drop-empty="true" rename="destinationExpansionDevice_
destinationExpansionDevice" />
        <topology group="Core_Networking" destinationType="DisplayClassName" relation="
HostedBy" reverse="true"/>
    </data>
</data>

```

The previous example will translates to the following (The id is generated randomly) :

```

<!-- SNMPAgent HostedBy/HostsServices Router/Switch/Host/FileServer (reverse) -->
<data class="SNMPAgent" containment="HostedBy" start="true" id="generated-id">
    <property name="DisplayName.Name" drop-empty="true" rename="sourceExpansionPart_
sourceExpansionPart"/>
    <property name="DisplayClassName" rename="_sourceType" />
</data>

<data class="Router_Switch_Host_FileServer" from="generated-id">
    <property name="DisplayName.Name" drop-empty="true" rename="sourceExpansionDevice_
sourceExpansionDevice" />
    <property name="DisplayName.Name" drop-empty="true" rename="destinationExpansionDevice_
destinationExpansionDevice" />
    <topology group="Core_Networking" destinationType="DisplayClassName" relation="HostedBy"
reverse="true"/>
</data>

```


Examples

This section explains a few examples and means to prevent pitfalls.

1 Example1 : Direct relation

Let's say we have the following objects in our topology definitions:

- The topology group we want to add objects to is 'Core Networking'.
- We have a SNMPAgent type in our 'Core Networking' group.
- The previous type is defined by the following expansion : device,part
- We also have a type names Router, Switch, Host and FileServer.
- The previous four types are defined by the following expansion : device
- SNMPAgent has a relation named 'HostedBy' with the four previous types.
- All four relations between SNMPAgent and the four previous types have a reverse relation named 'HostsServices'.

Knowing all the previous topology information, we can now proceed to create the configuration chunk to retrieve those relations from SMARTS.

```

<!-- SNMPAgent HostedBy/HostsServices Router/Switch/Host/FileServer (reverse) -->
<data class="SNMPAgent" containment="HostedBy" start="true">
  <property name="DisplayName_Name" drop-empty="true" rename="sourceExpansionPart_
sourceExpansionPart"/>
  <property name="DisplayClassName" rename="_sourceType" />

  <data class="Router_Switch_Host_FileServer">
    <property name="DisplayName_Name" drop-empty="true" rename="sourceExpansionDevice_
sourceExpansionDevice" />
    <property name="DisplayName_Name" drop-empty="true" rename="destinationExpansionDevice_
destinationExpansionDevice" />
    <topology group="Core_Networking" destinationType="DisplayClassName" relation="
HostedBy" reverse="true"/>
  </data>
</data>

```

2 Example2 : NetworkConnection fork branching

Let's say we have the following objects in our topology definitions:

- The topology group we want to add objects to is 'Core Networking'.
- We have a Router type in our 'Core Networking' group.
- The previous type is defined by the following expansion : device
- Router has a relation named 'IsLinkedTo' with another Router

Knowing all the previous topology information, we can now proceed to create the configuration chunk to retrieve those relations from SMARTS.

```

<script id="prevent-already-seen-navigation">
  <![CDATA[
    !pctx.hasSeenInstance(smInstance);
  ]]>
</script>

<!-- Router -> ( Interface -> NC -> Interface ) -> Router -->
<data id="R1" class="Router" containment="ComposedOf" start="true">
  <property name="DisplayName_Name" drop-empty="true" rename="sourceExpansionDevice_
sourceExpansionDevice" />
  <data class="Interface" containment="ConnectedVia">

```

```

        <data class="NetworkConnection" containment="ConnectedTo">
          <data class="Interface" containment="PartOf">
            <pre-filter-script script-id="prevent-already-seen-navigation"/>
            <data class="Router">
              <property name="DisplayName_Name" drop-empty="true" rename="
destinationExpansionDevice_destinationExpansionDevice"/>
              <topology group="Core_Networking" sourceType="Router" destinationType="
Router" relation="IsLinkedTo"/>
            </data>
          </data>
        </data>
      </data>
    </data>
  </data>

```

This fetching will navigate through the following containments:

- 1.Router ->ComposedOf ->Interface
- 2.Interface ->ConnectedVia ->NetworkConnection
- 3.NetworkConnection ->ConnectedTo ->Interface1,Interface2
- 4.Interface[12] ->PartOf ->Router

If you look carefully at step 3, the NetworkConnection will fork to 2 interfaces, one of which we are coming from. To prevent the creation of Router1 ->Interface1 ->NetworkConnection ->Interface1 ->Router1 (creating a link with ourself), we can use a pre filter script using the hasSeenInstance method to see if we have not already passed from the current instance relation. With this filter, the topology links created will be Router1 ->Interface1 ->NetworkConnection ->Interface2 ->Router2 and Router2 ->Interface2 ->NetworkConnection ->Interface1 ->Router1

Topology properties

The following properties are used by the internal engine object topology handler. If any of the mandatory options are not specified, the creation of the topology link is skipped.

- `_[source|destination]Type` defines the name of the source or destination type.
- `_[source|destination]Filter(optional)` defines the additional filter for the source or destination instance.
- `_group` defines the name of the topology group this internal object applies to.
- `_relation` defines the name of the topology relation this internal object applies to.
- `_reverse(optional, defaults to false)` defines if the handler needs to create the reverse relation between the source and destination instance.