HP Service Quality Management solution

HP SQM Solution Generic DDP

V3.0 Integration Guide

Edition: 2.0

for Microsoft Windows 64-bit Operating System

Nov 2011

Legal Notices

Warranty

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

License Requirement and U.S. Government Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Microsoft®, Windows® and Windows NT® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

Origin

Printed in English.

Contents

Preface

This document provides reference information to help configure and use Generic Discovery & Data load Pack (Generic DDP) on Service Management Foundation.

Purpose of **Generic Discovery & Data load Pack** (**DDP**) is to discover CIs and relationships based on information retrieved from the following sources:

- External 3PP database
- External function which return the data source(provided by user)
- XML files
- External function which return the data source(provided by TeMIP)

And create these discovered objects and save to the CMDB.

This document describes how to:

- Configure the relevant settings of Discovery job and Run it
- Modify the xml file

Intended Audience

This document is aimed at the following personnel:

- SQM Solution Delivery Engineer
- Developer for SQM Adapter

It is assumed that the readers have got a brief of Discovery and Dependency Mapping.

Please read books in Associated Documents for reference if you have some questions.

Software versions

Name	Version	Operating System	Description
Generic Discovery & Data load Pack	3.0	Windows 2003 or 2008 64-bit	

Abbreviations and Acronyms

Table 1 List of Abbreviations and Acronyms

Term	Meaning
SQM	Service Quality Manager
SMF	Service Management Foundation

BSM	Business Service Management
UCMDB	Universal Configuration Management Data Base
SIS	Site Scope
KPI	Key Performance Indicator
MA	Monitoring Adapter
DDM	Discovery and Dependency Mapping
DDP	Discovery & Data load Pack
KES	KPI Enrichment Service
CIT	Configuration Item Type
CI	Configuration Item
SLM	Service Level Management

Associated Documents

- *HP Discovery and Dependency Mapping*

The HP Business Availability Center and Discovery Dependency Mapping documents are available at:

http://support.openview.hp.com/selfsolve/manuals

- *sqm_service_management_foundation_installation_and_configuration_guide.doc*

Additional SQM Solution materials (like the SQM Solution product briefs) and information about SQM Solution updates are available at:

http://www.hp.com/cms

→ Section "Next Generation Operations Support System Solutions"

→ Section "Assurance"

→ Section “HP Service Quality Management Solution”

→ Section "Learn More"

Support

You can visit the HP Software support web site at:

http://www.hp.com/go/hpsoftwaresupport

This Web site provides contact information and details about the products, services, and supports that HP Software offers.

HP Software online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit enhancement requests online
- Download software patches
- Submit and track progress on support cases
- Manage a support contract

- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Chapter 1

Introduction for Generic DDP

Generic Discovery & Data load Pack (**DDP**) is to discover CIs and relationships based on information retrieved from the following sources:

- External 3PP database
- External function which return the data source(provided by user)
- XML files
- External function which return the data source(provided by TeMIP)

And create these discovered objects and save to the CMDB.

It has five packages to serve its purpose:

- SQM_DDP_CommonLib
- SQM_DDP_from_Database
- SQM_DDP_from_XML
- SQM_DDP_from_TeMIP
- SQM_DDP_Lib

They can be installed into BSM after executing *SQMSolSMFV300RevC.exe*.

1.1 SQM_DDP_CommonLib

This package contains Jython script files and jar files as external resource which could be integrated into other application to load CIs and CI relationships.

1.2 SQM_DDP_from_Database

This package provides a whole discovery template for creating CIs and relationships from a specified database, including a module, a job, a pattern and a script.

- *in_db_discovery.py*

  Provide the main function which will be executed by Probe.

  This script imports some scripts from SQM_DDP_CommonLib to achieve the target.

1.3 SQM_DDP_from_XML

This package provides a whole discovery template for creating CIs and relationships from specified xml files, including a module, a job, a pattern and one script.

- *in_xml_discovery.py*

  Provide the main function which will be executed by Probe.

  This script imports some scripts from SQM_DDP_CommonLib to achieve the target.

1.4 SQM_DDP_from_TeMIP

This package provides a whole discovery template for creating CIs and relationships from specified TeMIP Data, including a module, a job, a pattern and one script.

- *BAC_Loader_launch.py*

  Provide the main function which will be executed by Probe.

  This script imports some scripts from SQM_DDP_CommonLib to achieve the target.

1.5 SQM_DDP_Lib

This package provides jar files to support DDP to run.

1.6 More detail

For more detail, please check the following table:

Table 2 List of DDP content

Package	Content	Detail
SQM_DDP_CommonLib	Scripts	create_ci_from_db.py
		create_relationship_from_db.py
		generic_discovery_lib.py
		generic_xml_lib.py
		xml_lib.py
		dataload_sdk.py
	Ext Resrc	sqm-discovery.jar
		xmlbean.jar
		sqm_genericdiscovery_db.dtd
		sqm_genericdiscovery_xml.dtd
	Conf Files	log.properties
SQM_DDP_from_Database	Module	SQM_DDP_from_Database
	Job	SQM_DDP_from_Database
	Pattern	SQM_DDP_from_Database
	Scripts	in_db_discovery.py

SQM_DDP_from_XML	Module	SQM_DDP_from_XML
	Job	SQM_DDP_from_XML
	Pattern	SQM_DDP_from_XML
	Scripts	in_xml_discovery.py
SQM_DDP_from_TeMIP	Module	SQM_DDP_from_TeMIP
	Job	SQM_DDP_from_TeMIP
	Pattern	SQM_DDP_from_TeMIP
	Scripts	BAC_Loader_launch.py
SQM_DDP_Lib	Ext Resrc	apache.jar
		other.jar

1.7 Others

- *sqm_genericdiscovery_xml.dtd*

  This is the DTD file for the input XML file and is used by inXML-discovery to check the xml file to see whether the file is well formed or not.

- *sqm_genericdiscovery_db.dtd*

  This is the DTD file for the input XML file and is used by inDB-discovery to check the xml file to see whether the file is well formed or not.

The two DTD files are deployed into BAC server as DDM external resource. They will be automatically downloaded into DDM's client at the following path when DDM probe connects to BAC server.

```
<Probe's Installation
Path>\runtime\probeGateway\discoveryResources
```

Chapter 2

Deployment

This chapter explains how to deploy the package into the BSM. Once you finished deployment, you can follow the instructions in the next chapter to configure.

2.1 Software and Hardware Requirements

2.1.1 Software requirements

Table 3 Software Requirements

Software	Version
Microsoft Windows	2003 Server or later
HP Service Management Foundation	3.0
HP Business Service Management	9.1.X HP BSM 9.1.X(with an advanced license to use Discovery)
HP Discovery Probe	9.0.3
Database with the needed data	Oracle 10g or later
	MySQL 5.1.23

2.1.2 Hardware requirements

- CPU: Pentium IV 2.4 GHz or later
- Memory:1 GB RAM(min)
- Free hard disk space:8GB

For the complete hardware requirements, see the *HP Discovery and Dependency Mapping User Guide*.

2.2 Required Deployment

2.2.1 Import Generic DDP

Using SMF installer to deploy automatically

All the parts of DDP will be installed in BSM after SMF installer has been executed successfully.

Please read *HP SQM Solution Service Management Foundation Installation and Configuration Guide.doc* for the installation guide.

2.2.2 Verify the result of deployment

1. SQM_DDP_CommonLib

   Admin > RTSM Administration > Data Flow Management > Adapter Management

   Confirm that there is a package named "SQM_DDP_CommonLib" and contains all the following scripts.

   - *generic_discovery_lib.py*
   - *generic_xml_lib.py*
   - *create_ci_from_db.py*
   - *create_relationship_from_db.py*
   - *dataload_sdk.py*
   - *xmllib.py*

   Also its external resource should contain:

   - *sqm-discovery.jar*
   - *xmlbean.jar*
   - *sqm_genericdiscovery_db.dtd*
   - *sqm_genericdiscovery_xml.dtd*

   Besides it has a configuration file:

   - *log.properties*

2. SQM_DDP_from_Database

   Admin > RTSM Administration > Data Flow Management > Adapter Management

   There should be a package named SQM_DDP_from_Database which contains a pattern "SQM_DDP_from_Database" and a script:

   - *in_db_discovery.py*

   Admin > RTSM Administration > Data Flow Management > Discovery Control Panel

   There should be a module named SQM_DDP_from_Database and a job "SQM_DDP_from_Database"

3. SQM_DDP_from_XML

   Admin > RTSM Administration > Data Flow Management > Adapter Management

There should be a package named SQM_DDP_from_XML which contains a pattern “SQM_DDP_from_XML” and a script:

- *in_xml_discovery.py*.

Admin > RTSM Administration > Data Flow Management > Discovery Control Panel

There should be a module named SQM_DDP_from_XML and contains a job “SQM_DDP_from_XML”

4. SQM_DDP_from_TeMIP

   Admin > RTSM Administration > Data Flow Management > Adapter Management

   There should be a package named SQM_DDP_from_TeMIP which contains a pattern “SQM_DDP_from_TeMIP” and a script:

   - *BAC_Loader_launch.py*.

   Admin > RTSM Administration > Data Flow Management > Discovery Control Panel

   There should be a module named SQM_DDP_from_TeMIP and contains a job “SQM_DDP_from_TeMIP”

5. SQM_DDP_Lib

   Admin > RTSM Administration > Data Flow Management > Adapter Management

   There should be a package named SQM_DDP_Lib which contains:

   - *apache.jar*
   - *other.jar*

2.2.3 Install UCMDB Dataflow Probe 9.0.3

See *DiscoveryDependencyMapping.pdf* for installation guide.

Just remember to install Probe Gateway and Management together (not in separated mode).

2.2.4 Place the XML file

The XML files which DDP uses should be saved to any path at the PC where the Probe is installed.

See 4.1 to learn more about how to refer to DTD in XML file

Chapter 3

Simple Configure and Run

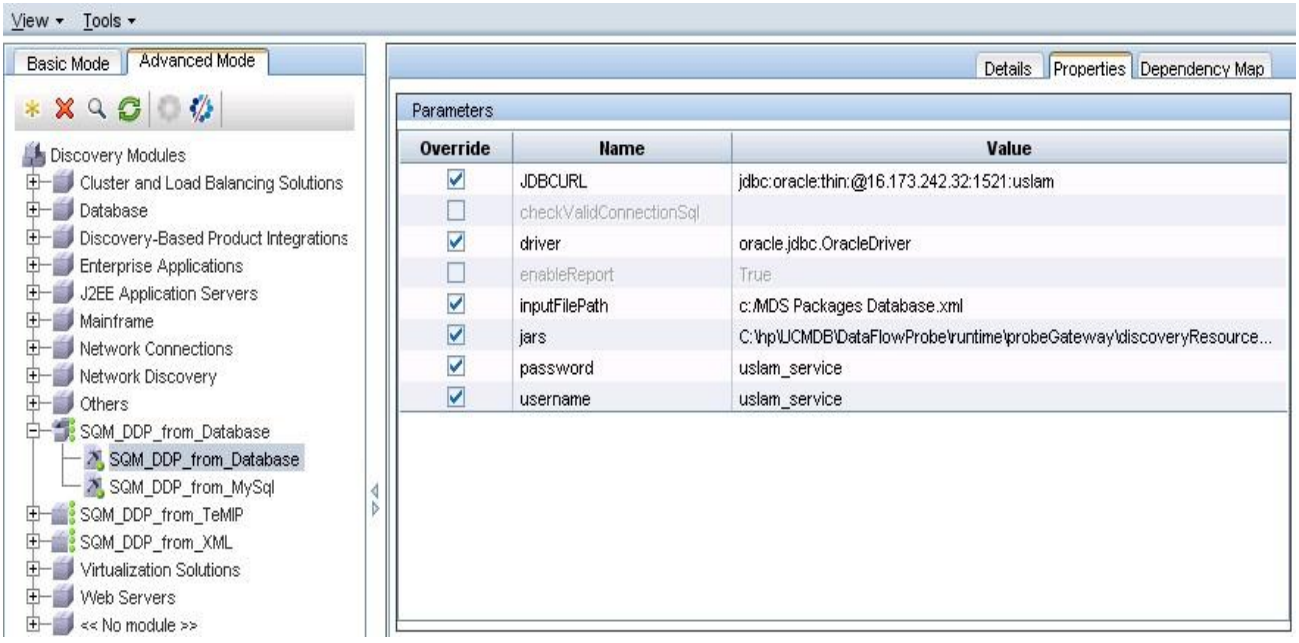
This chapter described how to configure the discovery settings of DDP and how to run it.

3.1 Simple Configuration

3.1.1 SQM_DDP_from_Database

1. Admin > RTSM Administration > Data Flow Management > Discovery Control Panel
2. Select the Job “SQM_DDP_from_Database” below SQM_DDP_from_Database in the Discovery Job Pane.
3. Click “Properties” tab in the right pane and edit “Parameters”.

Figure 5 Model of demo (CI part)



- *JDBCURL* required

  The URL for the DB connection

  E.G.

  ```
  jdbc:oracle:thin:@ibis.chn.hp.com:1521:sqlsa
  ```

- *username* required

Account to log into database

- *password* *r*equired

  The one used to log into database

- *jars* *r*equired

  Jar files which will be used as external resource especially for database driver.

  Multi jar files can be specified here. Use ";" to separate each other.

  The Jar file path must be full path

  E.G.

```
C:\hp\UCMDB\DataFlowProbe\runtime\probeGateway\discoveryResources\ojdb
C:\hp\UCMDB\DataFlowProbe\runtime\probeGateway\discoveryResources\mysq
```

  How to load external resource? Refer to step 5.

- *driver* *required*

  DB driver used

  E.G.

```
com.mysql.jdbc.Driver

oracle.jdbc.OracleDriver
```

- *inputFilepath* *required*

  This field is used to specify XML resource.

  It might be a single file, a folder or a ZIP file.

  If you want to specify file or folder, use absolute path to refer.

  As to ZIP file, you can specify it with absolute path or relevant path.

- *checkValidConnectionSql* *optional*

  A SQL statement to ensure that db-connection is available or not.

  It'll be executed before querying data each time.

- *enableReport* *required*

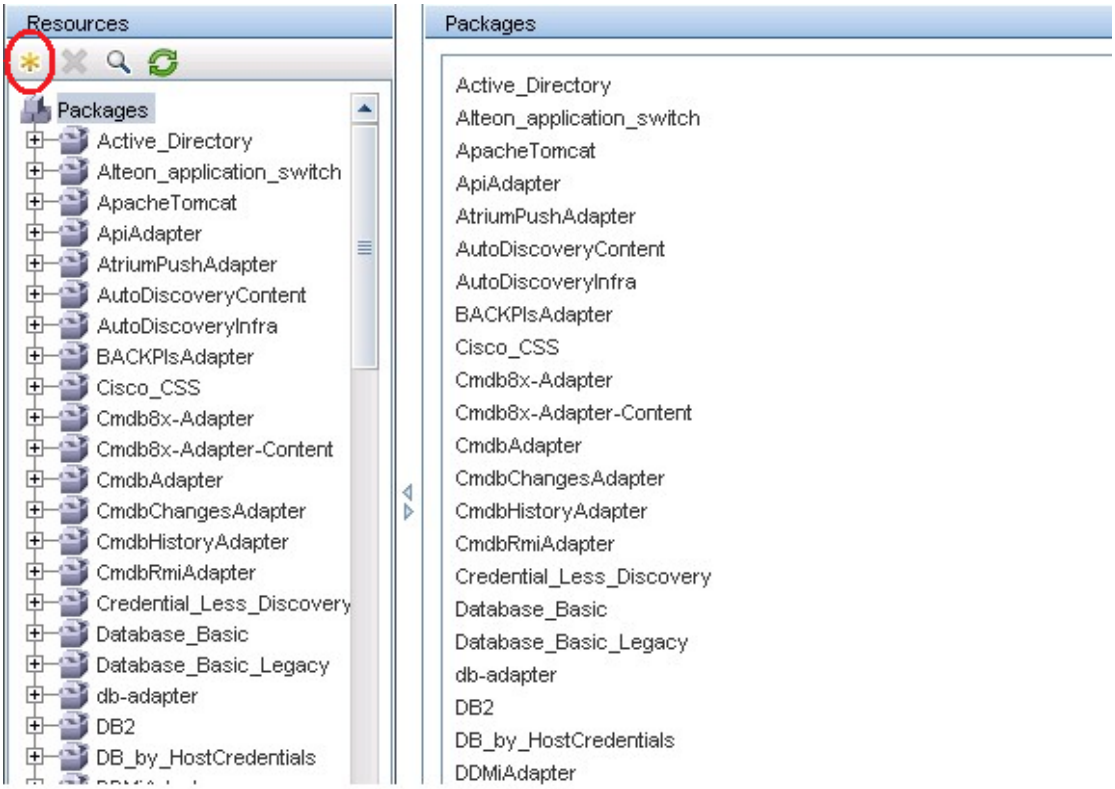
  True/False

  Enable report function or not

4. Admin > RTSM Administration > Data Flow Management > Adapter Management
5. Load the db driver to BSM as discovery external resource.

   Click the button ***** shown in Figure 6 and choose "Import external resource". Select the jar file needed and click ok.

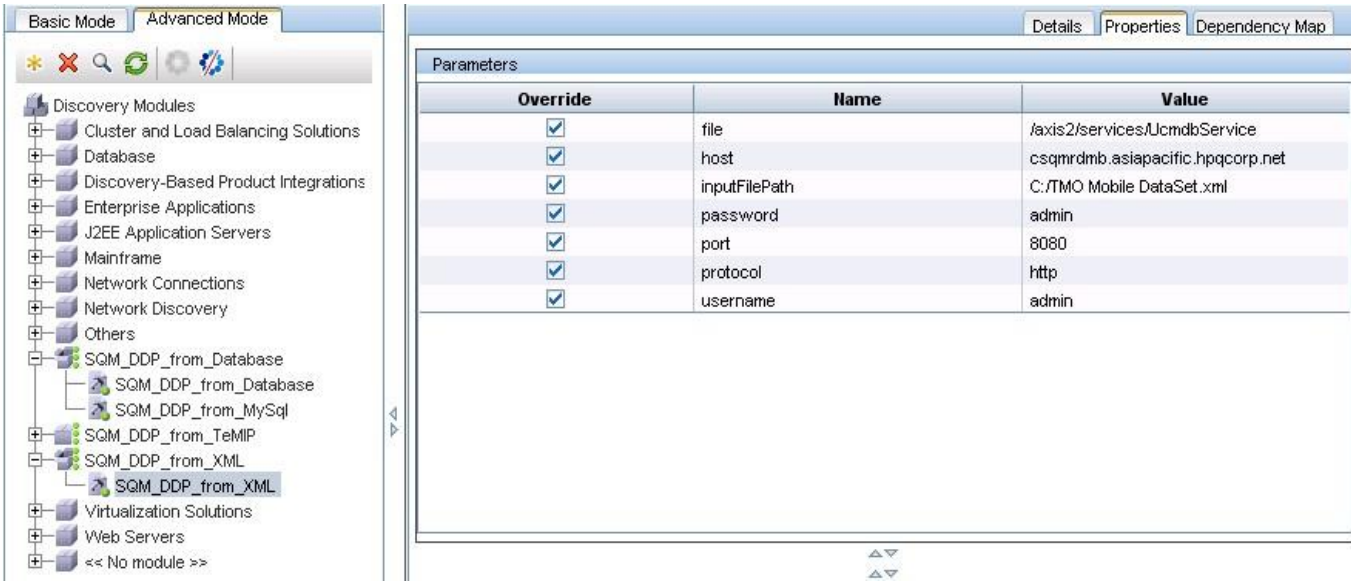
Figure 6 Load external resource



3.1.2 SQM_DDP_from_XML

1. Admin > RTSM Administration > Data Flow Management > Discovery Control Panel
2. Select the Job “SQM_DDP_from_XML” below shown in Figure 7 SQM_DDP_from_XML in the Discovery Control Panel.

Figure 7 Model of xml demo (CI part)



3. Click “attribute” tab in the right pane and edit “Parameters”

- *inputFilePath* *required*

  This field is used to specify XML resource.

  It might be a single file, a folder or a ZIP file.

  If you want to specify file or folder, use absolute path to refer.

  As to ZIP file, you can specify it with absolute path or relevant path.

- *username* *required*

  Account to log into BSM server JMX console

- *password* *required*

  Password is used to log on BSM server JMX console if necessary

- *protocol* *required*

  The protocol used to connect to the web service

  Only http is supported for now

- *host* *required*

  Hostname or IP of the web service host (BSM Gateway)

- *port* *required*

  BSM server JMX console’s port (e.g. 8080).

- *file* *required*

  The path of the WSDL file

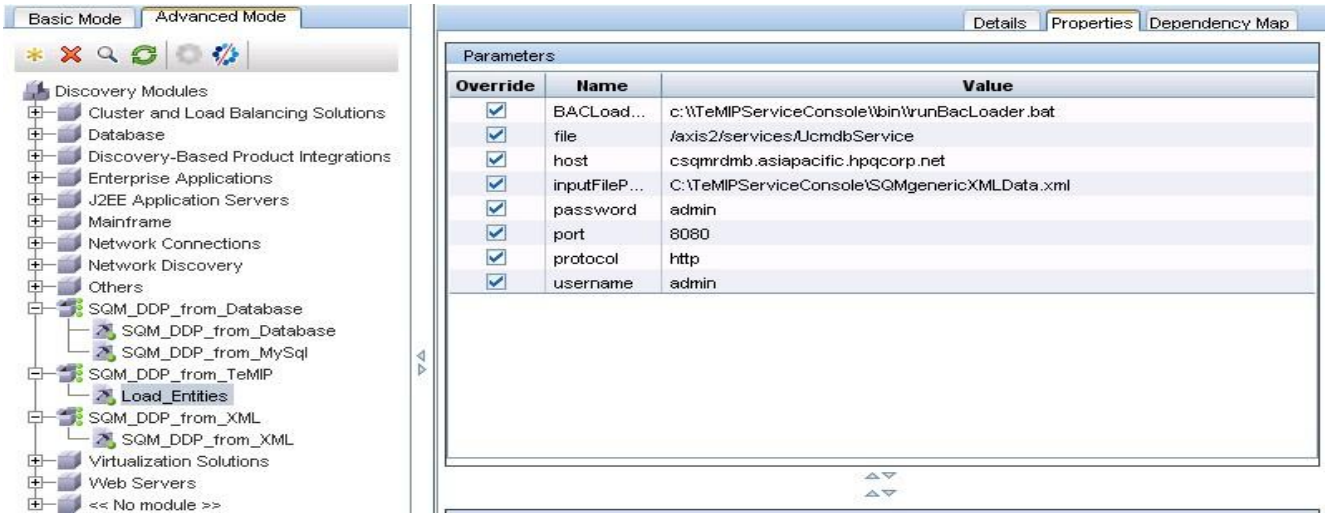
  The default setting for BSM is:

```
/axis2/services/UcmdbService
```

3.1.3 SQM_DDP_from_TeMIP

1. Admin > RTSM Administration > Data Flow Management > Discovery Control Panel
2. Select the Job “SQM_DDP_from_TeMIP” below shown in Figure 8 SQM_DDP_from_TeMIP in the Discovery Control Panel.

Figure 8 Model of TeMIP demo (CI part)



4. Click “attribute” tab in the right pane and edit “Parameters”

   - *inputFilePath* *required*

     This field is used to specify XML resource.

     It might be a single file, a folder or a ZIP file.

     If you want to specify file or folder, use absolute path to refer.

     As to ZIP file, you can specify it with absolute path or relevant path.

   - *username* *required*

     Account to log into BSM server JMX console

   - *password* *required*

     Password is used to log on BSM server JMX console if necessary

   - *protocol* *required*

     The protocol used to connect to the web service

     Only http is supported for now

   - *host* *required*

     Hostname or IP of the web service host (BSM Gateway)

   - *port* *required*

     BSM server JMX console’s port (e.g. 8080).

   - *BACLoaderPath* *required*

     TeMIPServiceConsole’s batch file load from TeMIP server.

     c:\\TeMIPServiceConsole\\bin\\runBacLoader.bat

   - *file* *required*

     The path of the WSDL file

     The default setting for BSM is:

     `/axis2/services/UcmdbService`

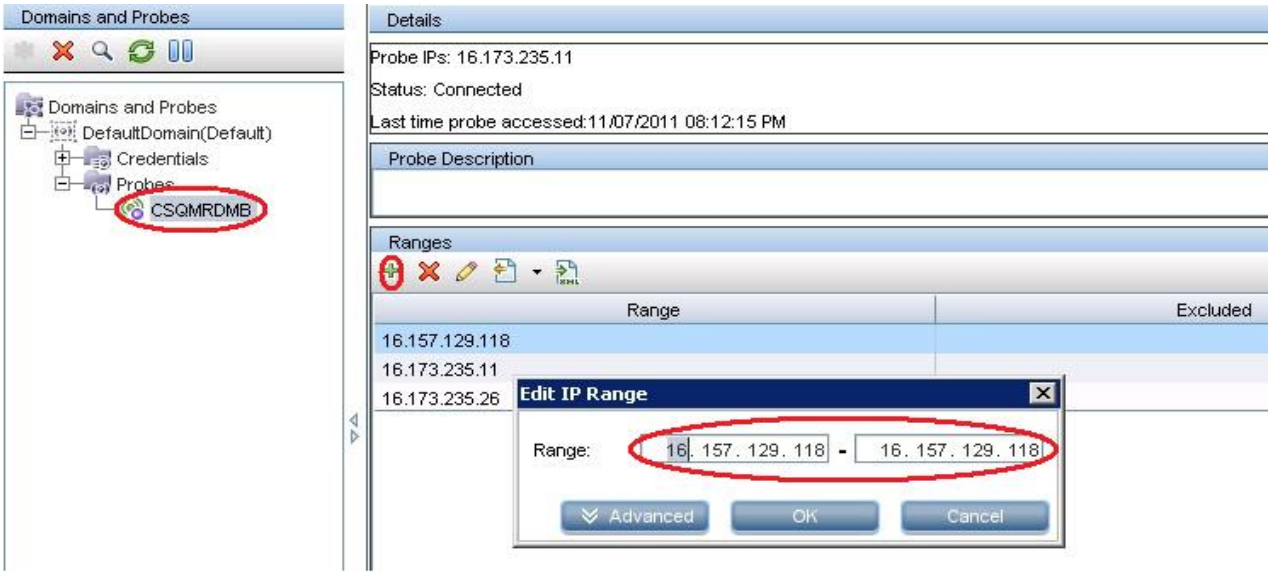
3.1.4 Run

All the patterns in this prototype have the same trigger CIT: “Discovery Probe Gateway”. CIs of this type can be automatically created when the probe is connected to BSM server. Please specify the IP range for the Probe and make it available to drive the job run.

Admin > RTSM Administration > Data Flow Management > Data Flow Probe Set up

1. Click the Probe which you use at the left pane.
2. Click the “+” icon at the right pane and specify IP range in the popup box.

Figure 9 Set valid IP range for Probe



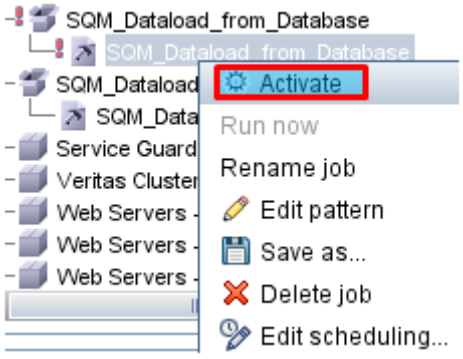
However, you may change the trigger CIT to whatever is available at your convenience.

Admin > RTSM Administration > Data Flow Management > Discovery Control Panel

Select the job, “SQM_DDP_from_Database” under the module “SQM_DDP_from_Database”.

1. Right-click it and choose "Activate" on the pop menu

Figure 10 Activate Discovery Job



2. Do the same thing to the job “SQM_DDP_from_XML” and “SQM_DDP_from_TeMIP”
3. Wait a moment and press the "refresh" ICON to see result.

   Check the result pane and see whether CI or link is discovered or not

Chapter 4

Advanced Usage

Advanced settings of the packages will be introduced.

Besides, we'll give a brief introduction of the workflow to help user analyze the errors which might occur.

4.1 Referring to DTD file

DDP uses XML files as its configuration file.

These XML files must be written in compliance with DDP's DTD files.

- *sqm_genericdiscovery_xml.dtd*

  Definition for XML files used by SQM_DDP_from_XML

- *sqm_genericdiscovery_database.dtd*

  Definition for XML files used by SQM_DDP_from_Database

4.1.1 Absolute Path Referring

Use the DTD file's absolute path to refer. For example:

```
<!DOCTYPE entities SYSTEM "
C:\hp\UCMDB\DataFlowProbe\runtime\probeGateway\discoveryResources\sqm_ger
```

4.1.2 Relevant Path Referring

Use the DTD file's relevant path to refer.

Example 1: While DTD file and XML file are in the same folder

```
<!DOCTYPE entities SYSTEM " sqm_genericdiscovery_xml.dtd">
```

Example 2: While DTD file is in the upper folder of XML file

```
<!DOCTYPE entities SYSTEM " ..\sqm_genericdiscovery_xml.dtd">
```

If XML files are packed into ZIP file to deploy, these file will be extracted into the following path

```
<Probe's Installation
Path>\runtime\probeGateway\discoveryResources\<ZIP file
name>\
```

Meanwhile, DTD file will be automatically downloaded into DDM Probe's client at:

```
<Probe's Installation Path>\runtime
\probeGateway\discoveryResources
```

In this case, use "..\sqm_genericdiscovery_xml.dtd" to refer to the upper folder's DTD file.

4.2 SQM_DDP_from_Database

The xml file which describes the definition of CIs and relationships will be introduced in this sector.

4.2.1 CI Part

Let's see an example first:

```
<CIT CItype="BSO_POC_Stream">
      <selector><![CDATA[select * from BSO ]]></selector>
            <keyAttributes>
                  <attribute name="sid_object_identifier"
dataType="string" formatting="">
                        <columns>
                              <column name="sid"/>
                        </columns>
                  </attribute>
            </keyAttributes>
            <attributes>
                  <attribute name="display_label"
dataType="string" formatting="">
                        <columns>
                              <column name="label"/>
                        </columns>
                  </attribute>
                  <attribute name="name"
dataType="string" formatting="">
                        <columns>
                              <column name="bsoname"/>
                        </columns>
                  </attribute>
                  <attribute name="bso_location"
dataType="location_type_enum" formatting="">
                        <columns>
                              <column name="location"/>
                        </columns>
                  </attribute>
                  <attribute name="bso_list"
dataType="string_list" formatting=",">
                        <columns>
                              <column name="groupone"/>
                              <column name="grouptwo"/>
                              <column name="groupthird"
                        </columns>
                  </attribute>
            </attributes>

</CIT>
```

This is a sample for the construction of the CI part.

1. <CIT>

   Each <CIT> represents one CI's definition.

Its attribute:

- CItype

  CIT name

- table

  Table from where the data is extracted.

  It also contains one <keyAttributes> and one <attributes>.

  - <keyAttributes>

    Key attribute(s) of the specified CIT

  - <attributes>

    Non-key attribute(s) of the specified CIT

  - <selector>

    Specify the data source with SQL statement.

    See 4.1.3 for more detail

  <keyAttributes> and <attributes> are almost the same, but <keyAttributes> must contain one <attribute> at least while <attributes> needn't.

2. <attribute>

   This element represents the information for converting DB's fields into CI's attributes.

   Its attribute:

   - name

     Attribute name

   - dataType

     Attribute data type

     The following types are supported now.

     - string
     - integer/ int
     - long
     - double
     - boolean/bool
     - string_list
     - location_type_enum

   - formatting

     Formatting function

     This is the way to combine <column>s' values to generate the attribute's value.

     It could be:

     - Blank

       Meaning no specific formatting function to be applied

     - <SEPARATOR>

The default formatting function with the separator string specified

- <customer_specified_function>

  Formatting function which is specified by the customer. The naming of it is:

  "function:" + Jython file name + "." + function name

  For example,

  ```
  function:customfunction.getUpper
  ```

  This means that the script will execute the function "getUpper" in *customfunction.py* to generate the current attribute's value.

  A sample for the function:

  ```
  def getLower(target):

      return str(target['key2']).lower()
  ```

  The function has only one parameter which is a python's dictionary. It contains the values of all the columns and the special parameter "Framework".

  ```
  target[<column name>]
  ```

  Get the given column value.

  ```
  target["framework"]
  ```

  Get framework passed by the discovery script.

- predefinedValue

  Default value for the current attribute's value

  If predinedValue is not null or blank string, the current attribute's value will be replaced with. (This means that contents of the columns will be ignored.)

3. <columns>

   Subject has several <column> elements in it and each <column> represents a field.

   <column> has only one attribute "name" which represents the field name in a certain table or view.

4.2.2 Relationship Part

Let's see an example first:

```
<link linkname="depends_on" table="V_PRODUCT_REL">
   <attributes>
      <attribute name="data_name" dataType="string"
formatting="" predefinedValue="testRelationshipAttr"/>
   </attributes>
   <originCI CItype="VFProduct">
      <keyAttributes>
         <attribute name="id" dataType="string"
formatting="">
         <columns>
            <column name="product_id"/>
```

```
            </columns>
         </attribute>
         </keyAttributes>
      </originCI>
      <endCI CItype="VFService">
         <keyAttributes>
            <attribute name="id" dataType="string"
formatting="">
               <columns>
                  <column name="service_id"/>
               </columns>
            </attribute>
         </keyAttributes>
      </endCI>

</link>
```

This is a sample for the construction of the relationship (also called link) part.

1. <link>

   Each <link> represents one kind of **valid link**'s definition.

   Pay attention that "depends_on" is just a type of relationship while depends_on: VFProduct → VFService means a valid link.

   Its attribute:

   - linkname

     Type name of the current valid link

   - table

     Table from where the data is extracted.

   It contains 4 tags:

   - <attributes>

     This represents the attributes of the relationship.

     Just like the one in <CIs> sector, it has <attribute> as its child and the structure is the same. Please read 4.1.1 for reference.

   - <originCI>

     This represents the origin point of the link

   - <endCI>

     This represents the end point of the link.

   - <selector>

     Specify the data source with SQL statement.

     See 4.1.3 for more detail

2. <originCI> and <endCI>

   Both <originCI> and <endCI> have an attribute "CItype" to show its CIT and an <keyAttributes> element.

   <keyAttribute> here is just the same as the one in the CI part. Please read 4.1.1 for reference.

4.2.3 Use <selector> to specify data source

As mentioned before, we can specify the attribute “table” to gain data source. Beside it, we have another choice to use <selector> instead.

Let’s see an example first:

```
<CIT CItype="test_cell">
  <selector><![CDATA[select * from
mms_test_v ]]></selector>
  <keyAttributes>
     <attribute name="data_name" dataType="string"
formatting="_">
       <columns>
         <column name="country_id"/>
         <column name="region_id"/>
         <column name="lai"/>
         <column name="cell_id"/>
       </columns>
     </attribute>
  </keyAttributes>
  <attributes>
    <attribute name="location_area_identity"
dataType="string"  formatting="">
      <columns>
        <column name="lai"/>
      </columns>
    </attribute>
    <attribute name="cell_identifier" dataType="string"
formatting="">
      <columns>
        <column name="cell_id"/>
      </columns>
    </attribute>
  </attributes>

</CIT>
```

The <selector> could be written in two patterns.

4.2.3.1 SQL statement

```
<selector><![CDATA[select * from mms_test_v]]></selector>
```

If the SQL statement includes some special XML character, the statement must be surrounded by a CDATA tag like the sample shown above.

Note

In an XML document or external parsed entity, a **CDATA** section is a section of element content that is marked for the parser to interpret as only character data, not markup. A CDATA section is merely an alternative syntax for expressing character data;

there is no semantic difference between character data that manifests as a CDATA section and character data that manifests as in the usual syntax in which "<" and "&" would be represented by "<" and "&", respectively.

The script will execute the SQL statement and get the result set as data source of CI or Link creation.

4.2.3.2 Common data source function.

Let’s see an example first:

```
<selector>function:MyFunction.myDataResource</selector>
```

If <selector> start with “function:” that means it is a data source function.

The name of a data source function could be specified by the customer.

Its naming rule is: “function:” + Jython file name + “.” + function name

```
function:custom_function.getDataResource
```

This means that the script will execute the function “getDataResource” in *custom_function.py* to get the result set as data source for CI or Link creation.

A sample for the function:

```
def getDSforTest(framework):
    logger.info("Inner application.");
    data = []
    data.append({'key':’1','booleancol':'true'})

    return data
```

The function has only one parameter “framework”.

This is an interface that can be used to retrieve information that is required to run the discovery, such as information on the trigger CI, pattern parameters, and is also used to report on errors that occur during running of the script. Please refer to the "Discovery SDK" for the Framework full API information.

The return value is a python list which contains a lot of data represented as a python dictionary.

The return value must meet the content of XML.

E.g. in above example xml, it contains four columns.

```
<column name="country_id"/>
<column name="region_id"/>
<column name="lai"/>
<column name="cell_id"/>
```

So the return value should like below

```
[{'cell_id':'40005'\
    ,'lai':'234.42.1',\
    'country_id':'AUSTRALIA',\
    'region_id':'Aachen'},\
    {'cell_id':'40009',\
    'lai':'234.42.1',\
    'country_id':'AUSTRALIA',\
    'region_id':'Aachen'}]
```

The inDB-discovery will take the returned value as data source of CI and link creation.

4.3 SQM_DDP_from_XML

The input XML file which represents the instances of CIT will be introduced in this sector.

4.3.1 CI Part

Let’s see an example first:

```
<CI CItype="BSO_POC_Stream" >
```

```
            <keyAttributes>
                  <attribute name="sid_object_identifier"
dataType="string" value="BSO_Voice1" formatting="" />
            </keyAttributes>
            <attributes>
                  <attribute name="display_label"
dataType="string" value="BSO Voice1" formatting="" />
                  <attribute name="name"
dataType="string" value="BSO Customer" formatting="" />
                  <attribute name="bso_location"
dataType="location_type_enum" value="city" formatting=""
/>
                  <attribute name="bso_list"
dataType="string_list" value="TEMIP,TSC,MYSQL"
formatting="" />
            </attributes>
</CI>
```

1. <CI>

   Each <CI> represents an instance of a certain CIT

   It has one attribute:

   - CItype

     CIT name

   There are two elements in it:

   - <keyAttributes>

     Key attribute(s) of CIT

   - <attributes>

     Non-key attribute(s) of CIT

   They are almost the same except that <keyAttributes> contains at least one <attribute> while <attributes> might has no <attribute> in it at all.

2. <attribute>

   This represents an attribute of a CI. It has 3 attributes.

   - name

     Attribute name

   - dataType

     The following types are supported by now.

     - string
     - bytes
     - boolean
     - integer
     - long
     - double
     - date
     - `string_list`
     - `location_type_enum`

- value

  Value of the attribute

  Be sure to fill it will a string.

- formatting

  This could be a blank string, a conjunction string or a function written in Jython.

  Use the string like “function:customer_function.getUpper” to specify formatting function.

```
def getUpper(target):
    return (str(target)).upper()
```

  This is a sample for formatting function in XML discovery part.

  The function has only one parameter to get the value defined in xml file. Its output is supposed to be a string.

4.3.2 Relationship Part

Let’s see an example first:

```
<link linkname="depends_on" weight="343.344">
  <originCI CItype="startCI">
    <keyAttribute name="key" dataType="string" value="B6654" />
    <keyAttribute name="key2" dataType="string" value="tail" />
  </originCI>
  <endCI CItype="endCI">
    <keyAttribute name="key" dataType="string" value="A6653" />
    <keyAttribute name="key2" dataType="string" value="peach"
/>
  </endCI>
</link>
```

This is a sample for the construction of the relationship(also called link) part.

1. <link>

   Each <link> here represents one instance of valid link.

   Note

   Be different from the ones in SQM_DDP_from_Database,

   Each <link> here represents **one instance** of valid link while <link> in SQM_DDP_from_Database represents **one kind** of valid link.

   Its attribute:

   - linkname

     Name of the current valid link

   Besides, it contains 3 tags:

   - <attributes>

     This represents the attributes of the relationship.

     Just like the one in <CIs> sector, it has <attribute> as its child and the structure is the same. Please read 4.1.1 for reference.

   - <originCI>

     This represents the origin point of the link

   - <endCI>

This represents the end point of the link.

Both <originCI> and <endCI> have an attribute “CItype” to show its CIT and an <keyAttributes> element.

The constructions of them are the same.

Both have an attribute “CItype” to show its CIT and an <keyAttributes> element.

2. <keyAttribute>

   This element is supposed to contain all key attributes of the CIT.

   Its construction is just like <attribute> in CI part.

   Note

   Be different from the ones in SQM_DDP_from_Database,

   User should list all the key attribute(s) of CIT so that only one CI instance or nothing could be matched in previous CI discovery step or in UCMDB. This ensures that only one link can be generated from one <link>.

   - name

     Attribute name

   - dataType

     Only the following types are supported by now.

     - string
     - boolean
     - integer
     - double

   - value

     Value of the attribute

     Be sure to fill it with a string.

   - formatting

     Formatting function definition

     See 4.2.1 for reference.

4.4 Workflow

Both of the two patterns we mentioned before have similar process:

1. Create CIs first
2. Create links

For the detail, please check this

Figure 8 Workflow for inDB-discovery

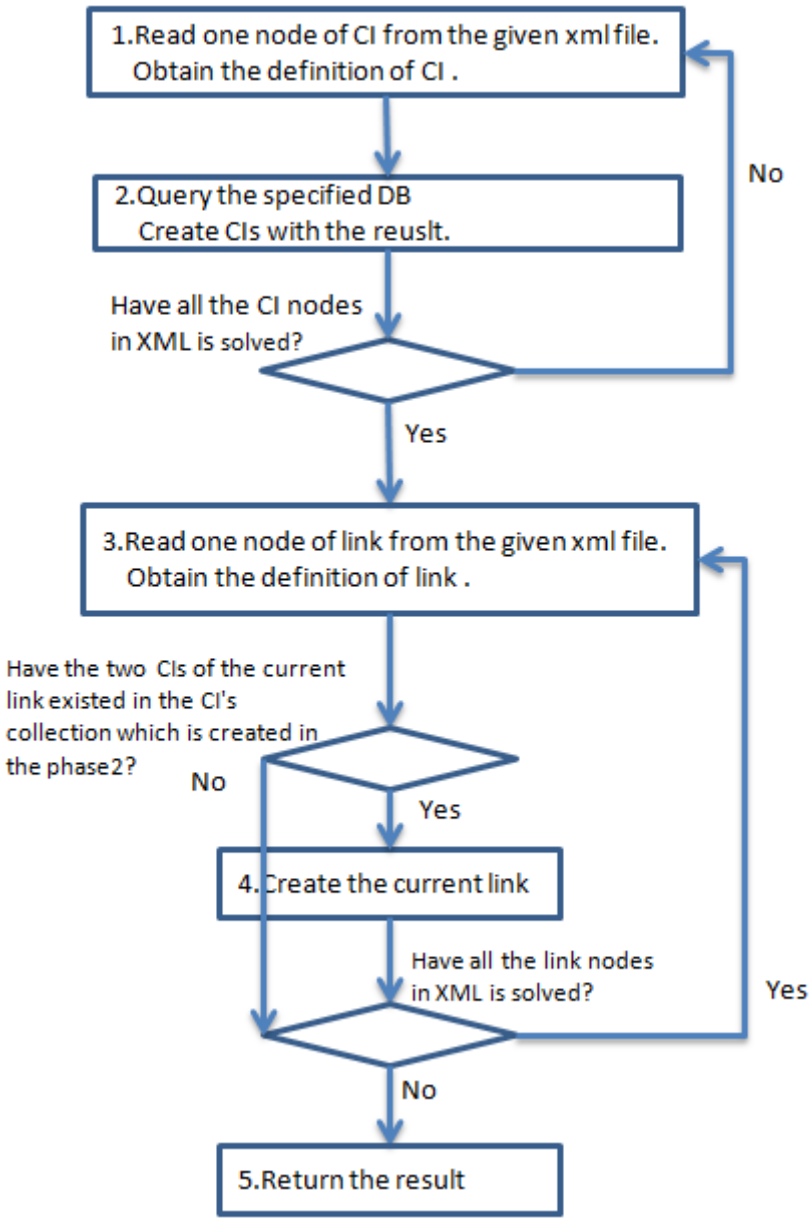
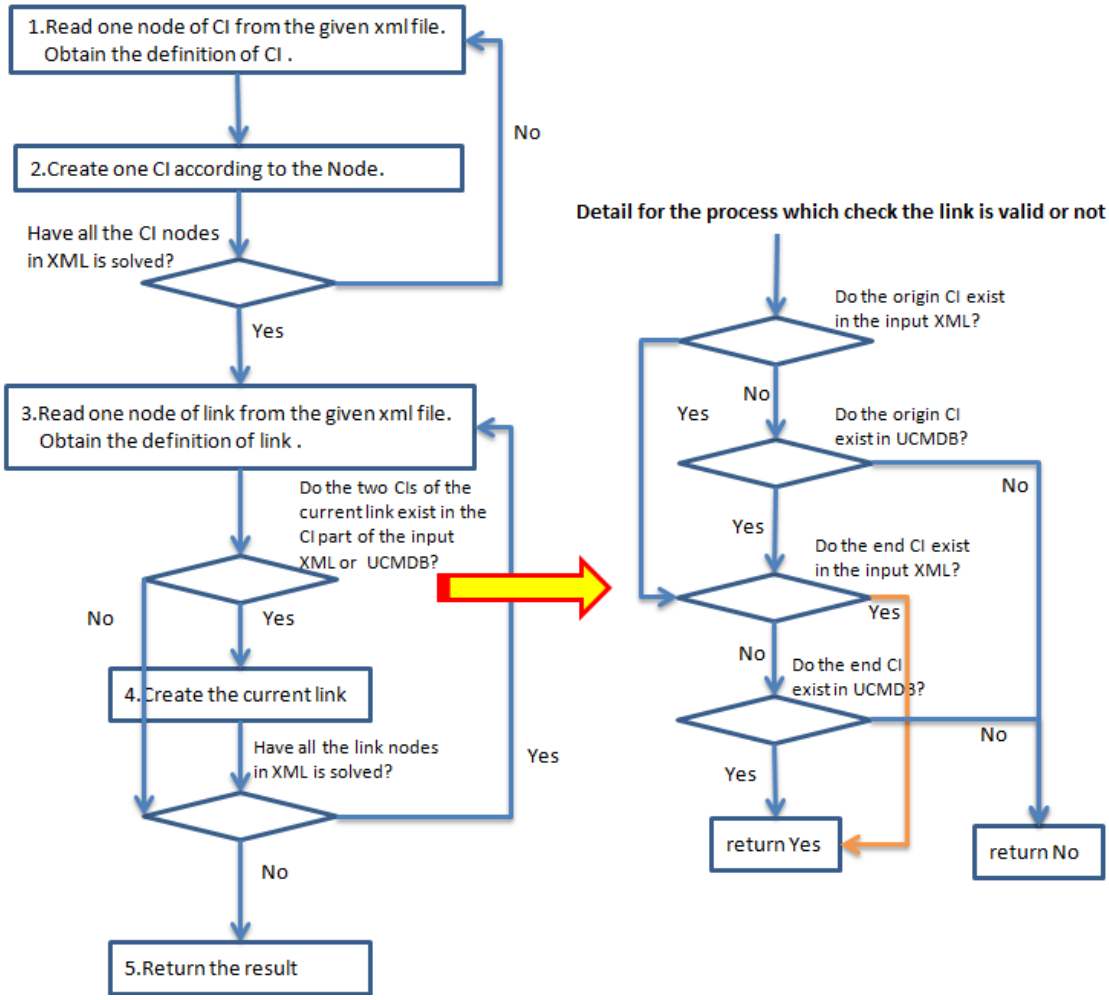

1.Read one node of CI from the given xml file.
Obtain the definition of CI .
2.Query the specified DB
Create CIs with the reuslt.
Have all the CI nodes
in XML is solved?
No
Yes
3.Read one node of link from the given xml file.
Obtain the definition of link .
Have the two CIs of the current
link existed in the CI's
collection which is created in
the phase2?
No
Yes
4.Create the current link
Have all the link nodes
in XML is solved?
Yes
No
5.Return the result

Figure 9 Workflow for inXML-discovery



.

4.5 Trouble shooting

4.5.1 Log file

We provide two log files to help to analyze.

Both of them are generated in the following path:

`<Probe Installation Path>\runtime\log`

- *sqm-genericDDP.log*

  Record the process of the DDP.

  All the invalid data sources will be recorded here.

  sqm-genericDDP-report.log

  Generate a new report for all the relationships and CIs created in this execution.

- *sqm-genericDDP-report.log*

Record the detail of discovery result generated by scripts.

Of course, the original log files for BSM and Discovery Probe still work. You can take them for reference too.

4.5.2 Load External Resource Jar Failure

Sometimes DDM is connecting to BAC server correctly, but external resource jars can't be recognized by DDM.

1. Check whether the jar has been transferred into DDM client at:

   ```
   <Probe's Installation
   Path>\runtime\probeGateway\discoveryResources
   ```

   If the jar does exist, then go to step 2.

   Otherwise, press Ctrl + C to terminate DDM probe, and restart it.

2. If the jar file exists, then press Ctrl + C to terminate DDM probe, and restart it.

   DDM probe is not so stable that jar file won't be included into the class path when it is downloaded for the first time.

4.5.3 Run a job immediately after activate it

Right-click the job and choose "Rerun discovery" from the popup menu.

Limitation

There are some limitations in both functional aspect and performance aspect:

4.6 Functional Limitation

- Not all CIT attribute types are supported for now,

  But below list of attributes are supported in DB and XML discovery.

  - For in DB_discovery
    - string
    - integer/ int
    - long
    - double
    - boolean/bool
    - string_list
    - `location_type_enum`
  - For in XML_discovery
    - string
    - bytes
    - boolean
    - integer
    - long
    - double
    - date
    - string_list
    - `location_type_enum`
- Not support multi databases yet

4.7 Performance

- Each job's executable time is 900 seconds by default, if the data processing takes too long, the job will be terminated forcedly and nothing will be reported to BSM.
- Discovery Probe's default heap is 256~512MB. This might be not enough for large scale data solving.

Glossary

This glossary contains definitions of terminology used in the SQM_Solution User Documentation set.

HP Business Service Management (BSM / HPBSM)

Refer to BSM associated documents in Preface Chapter

Service Management Foundation

Refer to SQM Solution associated documents in Preface Chapter

Discovery and Dependency Mapping (DDM)

The Discovery and Dependency Mapping (DDM) process is the mechanism that enables you to collect information about your system by discovering the IT infrastructure resources and their interdependencies. DDM automatically discovers and maps logical application assets in Layers 2 to 7 of the Open System Interconnection (OSI) Model.

Generic Dataload & Discovery Pack (Generic DDP)

Generic Discovery & Dataload Pack is a component of SQM-Solution which could discover CIs and links from the information extracted from database or xml file.

Discovery Probe

Discovery Probe is a component of DDM and acts as a data collector.

Probe will receive dispatched discovery job from BSM, execute it and return result to BSM.

Discovery Module

A set for several discovery jobs which have similar function.

Discovery Job

Discovery Jobs are instances of Discovery Pattern.

Discovery Pattern

One discovery pattern represents a certain pattern to discovery some data.

Discovery Script

These scripts are written in Jython 2.1 and serve the purpose of Discovery Pattern.