Tivoli Application Dependency Discovery Manager
Version 7 Release 2.1

SDK Developer's Guide

IBM
Tivoli Application Dependency Discovery Manager
Version 7 Release 2.1

SDK Developer's Guide

IBM
Note

Before using this information and the product it supports, read the information in "Notices" on page 141.
# Contents

**Tables** ............................................. v

**About this information** .......................... vii
Conventions used in this information ............ vii
Terms and definitions ................................. viii

**SDK Developer’s Guide** .............................. 1
Introducing the Software Developer’s Kit ........ 1
   Overview of the Software Developer Kit (SDK) .... 1
Introducing the Common Data Model ............... 1
Installing and configuring the Software Developer Kit 2
   System requirements ................................ 2
   TADDM SDK installation ............................. 2
   Configuring the TADDM SDK ........................ 4
   Verifying the SDK installation .................... 5
Using the TADDM SDK as a software component ... 6
SOAP API installation and configuration .......... 7
REST API installation and configuration .......... 7
Understanding the Common Data Model ............. 8
   Naming instances ................................. 12
   Class names ....................................... 15
   Dependencies between resources ................ 16
   Changes to the Common Data Model ............... 17

TADDM API overview .................................. 17
   Application programming interface overview .... 18
XML schema overview ................................ 18
JSON format overview ................................ 19
Model Query Language overview .................... 20
Using the Java API ................................... 24
Using the SOAP API ................................... 57
Developing applications using the REST API ....... 66
Command-line interface API ........................... 86
Developing custom server extensions ............... 94
   Overview .......................................... 95
   Managing extended attributes .................... 95
   Custom server extensions API .................... 96
TADDM database schema and views ................. 126
   Building block views .............................. 127
   Details pane views .............................. 129
   Custom views .................................... 133
   Extended attributes views ...................... 137
TADDM Data Dictionary ............................... 139

**Notices** ............................................. 141
   Trademarks ........................................ 143

© Copyright IBM Corp. 2006, 2014
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>TADDM SDK system requirements</td>
<td>2</td>
</tr>
<tr>
<td>2.</td>
<td>Embedded mode directory structure</td>
<td>3</td>
</tr>
<tr>
<td>3.</td>
<td>Standalone mode directory structure</td>
<td>3</td>
</tr>
<tr>
<td>4.</td>
<td>Configuration properties</td>
<td>4</td>
</tr>
<tr>
<td>5.</td>
<td>Changes to the Common Data Model for the TADDM release</td>
<td>17</td>
</tr>
<tr>
<td>6.</td>
<td>XML document structure</td>
<td>18</td>
</tr>
<tr>
<td>7.</td>
<td>MQL query elements</td>
<td>21</td>
</tr>
<tr>
<td>8.</td>
<td>MQL Operator Precedence</td>
<td>21</td>
</tr>
<tr>
<td>9.</td>
<td>Change history methods</td>
<td>30</td>
</tr>
<tr>
<td>10.</td>
<td>Discovery methods</td>
<td>32</td>
</tr>
<tr>
<td>11.</td>
<td>Find methods</td>
<td>33</td>
</tr>
<tr>
<td>12.</td>
<td>Management Software System methods</td>
<td>35</td>
</tr>
<tr>
<td>13.</td>
<td>MSSObjectLink</td>
<td>37</td>
</tr>
<tr>
<td>14.</td>
<td>Access list methods</td>
<td>38</td>
</tr>
<tr>
<td>15.</td>
<td>Collection methods</td>
<td>42</td>
</tr>
<tr>
<td>16.</td>
<td>Model management methods</td>
<td>42</td>
</tr>
<tr>
<td>17.</td>
<td>Relationship methods</td>
<td>48</td>
</tr>
<tr>
<td>18.</td>
<td>Session methods</td>
<td>48</td>
</tr>
<tr>
<td>19.</td>
<td>Version methods</td>
<td>50</td>
</tr>
<tr>
<td>20.</td>
<td>Metadata methods</td>
<td>51</td>
</tr>
<tr>
<td>21.</td>
<td>Presentation methods</td>
<td>51</td>
</tr>
<tr>
<td>22.</td>
<td>Security methods</td>
<td>53</td>
</tr>
<tr>
<td>23.</td>
<td>Application template methods</td>
<td>55</td>
</tr>
<tr>
<td>24.</td>
<td>Session requests</td>
<td>58</td>
</tr>
<tr>
<td>25.</td>
<td>Discovery requests</td>
<td>59</td>
</tr>
<tr>
<td>26.</td>
<td>Model and metadata requests</td>
<td>60</td>
</tr>
<tr>
<td>27.</td>
<td>Find requests</td>
<td>62</td>
</tr>
<tr>
<td>28.</td>
<td>Change history requests</td>
<td>64</td>
</tr>
<tr>
<td>29.</td>
<td>Version requests</td>
<td>65</td>
</tr>
<tr>
<td>30.</td>
<td>Extended attributes</td>
<td>96</td>
</tr>
<tr>
<td>31.</td>
<td>Capability functions</td>
<td>98</td>
</tr>
<tr>
<td>32.</td>
<td>Command and process functions</td>
<td>98</td>
</tr>
<tr>
<td>33.</td>
<td>Common Data Model functions</td>
<td>99</td>
</tr>
<tr>
<td>34.</td>
<td>DNS functions</td>
<td>99</td>
</tr>
<tr>
<td>35.</td>
<td>File access functions</td>
<td>100</td>
</tr>
<tr>
<td>36.</td>
<td>IP and MAC address functions</td>
<td>100</td>
</tr>
<tr>
<td>37.</td>
<td>Operating system functions</td>
<td>101</td>
</tr>
<tr>
<td>38.</td>
<td>Path functions</td>
<td>101</td>
</tr>
<tr>
<td>39.</td>
<td>Utility functions</td>
<td>102</td>
</tr>
<tr>
<td>40.</td>
<td>Version information functions</td>
<td>102</td>
</tr>
</tbody>
</table>
About this information

The purpose of this PDF document is to provide the related topics from the information center in a printable format.

The IBM Tivoli Application Dependency Discovery Manager Troubleshooting Guide and the troubleshooting topics in the information center include information on the following items:
- How to identify the source of a software problem
- How to gather diagnostic information, and what information to gather
- Where to get fixes
- Which knowledge bases to search
- How to contact IBM® Support

Conventions used in this information

This information describes the conventions that are used in the IBM Tivoli Application Dependency Discovery Manager (TADDM) documentation for denoting operating system-dependent variables and paths and for denoting the COLLATION_HOME directory. It also indicates the location of the collation.properties file, which is referenced throughout the TADDM documentation, including in the messages.

Operating system-dependent variables and paths

This information uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace $variable with %variable% for environment variables, and replace each forward slash (/) with a backslash (\) in directory paths.

If you are using the bash shell on a Windows system, you can use the UNIX conventions.

COLLATION_HOME directory

The COLLATION_HOME directory is the directory where TADDM is installed plus the dist subdirectory.

On operating systems such as AIX® or Linux, the default location for installing TADDM is the /opt/IBM/taddm directory. Therefore, in this case, the $COLLATION_HOME directory is /opt/IBM/taddm/dist.

On Windows operating systems, the default location for installing TADDM is the c:\IBM\taddm directory. Therefore, in this case, the %COLLATION_HOME% directory is c:\IBM\taddm\dist.
Location of collation.properties file

The collation.properties file contains TADDM server properties and includes comments about each of the properties. It is located in the $COLLATION_HOME/etc directory.

Terms and definitions

This information contains the terms and definitions for important concepts in the IBM Tivoli Application Dependency Discovery Manager (TADDM).

asynchronous discovery
In TADDM, the running of a discovery script on a target system to discover systems that cannot be accessed directly by the TADDM server. Because this discovery is performed manually, and separately from a typical credentialed discovery, it is called “asynchronous”.

business application
One or more computer programs or software components that provide functionality in direct support of a specific business process or processes.

business service
A group of diverse but interdependent applications and other system resources that interact to accomplish specific business functions.

CI
See configuration item.

collection
In TADDM, a group of configuration items.

configuration item (CI)
A component of IT infrastructure that is under the control of configuration management and is therefore subject to formal change control. Each CI in the TADDM database has a persistent object and change history associated with it. Examples of a CI are an operating system, an L2 interface, and a database buffer pool size.

credentialed discovery
TADDM sensor scanning that discovers detailed information about the following items:
• Each operating system in the runtime environment. This scanning is also known as Level 2 discovery, and it requires operating system credentials.
• The application infrastructure, deployed software components, physical servers, network devices, virtual systems, and host data that are used in the runtime environment. This scanning is also known as Level 3 discovery, and it requires both operating system credentials and application credentials.

credential-less discovery
TADDM sensor scanning that discovers basic information about the active computer systems in the runtime environment. This scanning is also known as Level 1 discovery, and it requires no credentials.

Data Management Portal
The TADDM web-based user interface for viewing and manipulating the data in a TADDM database. This user interface is applicable to a domain server deployment, to a synchronization server deployment, and to each storage server in a streaming server deployment. The user interface is very
similar in all deployments, although in a synchronization server
deployment, it has a few additional functions for adding and
synchronizing domains.

discover worker thread
In TADDM, a thread that runs sensors.

Discovery Management Console
The TADDM client user interface for managing discoveries. This console is
also known as the Product Console. It is applicable to a domain server
deployment and to discovery servers in a streaming server deployment.
The function of the console is the same in both of these deployments.

discovery server
A TADDM server that runs sensors in a streaming server deployment but
does not have its own database.

domain
In TADDM, a logical subset of the infrastructure of a company or other
organization. Domains can delineate organizational, functional, or
geographical boundaries.

domain server
A TADDM server that runs sensors in a domain server deployment and
has its own database.

domain server deployment
A TADDM deployment with one domain server. A domain server
deployment can be part of a synchronization server deployment.

In a domain server deployment, the following TADDM server property
must be set to the following value:
com.collation.cmdbmode=domain

launch in context
The concept of moving seamlessly from one Tivoli® product UI to another
Tivoli product UI (either in a different console or in the same console or
portal interface) with single sign-on and with the target UI in position at
the proper point for users to continue with their task.

multitenancy
In TADDM, the use by a service provider or IT vendor of one TADDM
installation to discover multiple customer environments. Also, the service
provider or IT vendor can see the data from all customer environments,
but within each customer environment, only the data that is specific to the
respective customer can be displayed in the user interface or viewed in
reports within that customer environment.

Product Console
See Discovery Management Console.

script-based discovery
In TADDM, the use, in a credentialed discovery, of the same sensor scripts
that sensors provide in support of asynchronous discovery.

SE
See server equivalent.

server equivalent (SE)
A representative unit of IT infrastructure, defined as a computer system
(with standard configurations, operating systems, network interfaces, and
storage interfaces) with installed server software (such as a database, a web
server, or an application server). The concept of a server equivalent also
includes the network, storage, and other subsystems that provide services to the optimal functioning of the server. A server equivalent depends on the operating system:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Approximate number of CIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>500</td>
</tr>
<tr>
<td>AIX</td>
<td>1000</td>
</tr>
<tr>
<td>Solaris</td>
<td>1000</td>
</tr>
<tr>
<td>Linux</td>
<td>1000</td>
</tr>
<tr>
<td>HP-UX</td>
<td>500</td>
</tr>
<tr>
<td>Network devices</td>
<td>1000</td>
</tr>
</tbody>
</table>

**storage server**

A TADDM server that processes discovery data that is received from the discovery servers and stores it in the TADDM database. The primary storage server both coordinates the discovery servers and all other storage servers and serves as a storage server. All storage servers that are not the primary are called secondary storage servers.

**streaming server deployment**

A TADDM deployment with a primary storage server and at least one discovery server. This type of deployment can also include one or more optional secondary storage servers. The primary storage server and secondary storage servers share a database. The discovery servers have no database.

In this type of deployment, discovery data flows in parallel from multiple discovery servers to the TADDM database.

In a streaming server deployment, the following TADDM server property must be set to one of the following values:

- com.collation.taddm.mode=DiscoveryServer
- com.collation.taddm.mode=StorageServer

For all servers except for the primary storage server, the following properties (for the host name and port number of the primary storage server) must also be set:

- com.collation.PrimaryStorageServer.host
- com.collation.PrimaryStorageServer.port

If the com.collation.taddm.mode property is set, the com.collation.cmdbmode property must not be set or must be commented out.

**synchronization server**

A TADDM server that synchronizes discovery data from all domain servers in the enterprise and has its own database. This server does not discover data directly.

**synchronization server deployment**

A TADDM deployment with a synchronization server and two or more domain server deployments, each of which has its own local database.

In this type of deployment, the synchronization server copies discovery data from multiple domain servers one domain at a time in a batched synchronization process.
In a synchronization server deployment, the following TADDM server property must be set to the following value:
com.collation.cmdbmode=enterprise

This type of deployment is obsolete. Therefore, in a new TADDM deployment where more than one server is needed, use the streaming server deployment. A synchronization server can be converted to become a primary storage server for a streaming server deployment. For more information, see Converting from a synchronization server deployment to a streaming server deployment.

**TADDM database**
In TADDM, the database where configuration data, dependencies, and change history are stored.

Each TADDM server, except for discovery servers and secondary storage servers, has its own database. Discovery servers have no database. Storage servers share the database of the primary storage server.

**TADDM server**
A generic term that can represent any of the following terms:
- domain server in a domain server deployment
- synchronization server in a synchronization server deployment
- discovery server in a streaming server deployment
- storage server (including the primary storage server) in a streaming server deployment

**target system**
In the TADDM discovery process, the system to be discovered.
SDK Developer's Guide

Introducing the Software Developer's Kit

This topic introduces the IBM Tivoli Application Dependency Discovery Manager (TADDM) Software Developer's Kit (SDK) and provides a brief overview of the TADDM Common Data Model.

The SDK Developer's Guide provides accurate visibility into business applications by providing application maps that highlight the relationship between the application and its supporting infrastructure. The comprehensive application maps include the infrastructure components that make up the application, their detailed configurations, and the runtime interrelationships and dependencies.

TADDM stores the topology data internally using a Java™ object hierarchy known as the Common Data Model (CDM).

Overview of the Software Developer Kit (SDK)

This SDK guide uses the open and scalable architecture of TADDM and provides you with a mechanism to quickly and efficiently reuse the comprehensive application maps across various application management solutions.

This SDK guide offers comprehensive access to the TADDM application maps and the discovery process, with which you can:

- Protect implementation investment by using a market proved, open, and standards-based integration SDK
- Ensure success of IT management initiatives by cost effectively sharing and reusing TADDM application maps across management applications
- Improve the accuracy of management solutions by integrating real-time and accurate application maps
- Use TADDM adapters and integrations for efficient deployments

The TADDM SDK provides a set of documented application programming interfaces (API):

- Java API
- SOAP API
- Representational State Transfer (REST) API
- Command-line interface (CLI) API

These APIs provide comprehensive access to TADDM application maps, including the discovered applications, their components, configurations, and dependencies. The APIs also offer complete control of the TADDM discovery process and its life cycle, including the starting, stopping, and managing of discoveries.

Introducing the Common Data Model

TADDM stores the topology data internally using a Java object hierarchy known as the Common Data Model.

The Common Data Model (CDM), which is persisted in a relational database, consists of model objects which represent discovered elements in the enterprise environment. The data model contains discovered objects of each element type,
such as computer systems or DB2® databases, with corresponding details represented as contained objects, such as operating systems or configuration values.

You can access the model using the IBM TADDM API, with all detail data displayed in the Data Management Portal accessible using this interface. The SDK represents data using an XML format with a published XML schema. Most contained objects are embedded within the document and objects that are referenced multiple times are duplicated within the document. The resulting XML document is somewhat larger than the original data, though easy to search using tools such as XQuery or XPath.

### Installing and configuring the Software Developer Kit

This topic describes the system requirements for using the IBM Tivoli Application Dependency Discovery Manager (TADDM) Software Developer's Kit (SDK) and explains how to install and configure the SDK.

### System requirements

This section describes the system requirements for using the TADDM SDK.

Table 1 lists the system elements and describes the respective requirement details.

<table>
<thead>
<tr>
<th>Element</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Any operating system that supports the Java 2 Runtime Environment (JRE)</td>
</tr>
<tr>
<td>Memory</td>
<td>1GB</td>
</tr>
<tr>
<td>Processors</td>
<td>1</td>
</tr>
<tr>
<td>Processor speed</td>
<td>1GHz</td>
</tr>
<tr>
<td>Disk space</td>
<td>200MB (Including the JVM)</td>
</tr>
<tr>
<td>Additional software requirements</td>
<td>Use TADDM 7.x Server JDK, version 5.0 if you are running the SDK on the same server with the TADDM server. The 5.0 JDK can be found in the $COLLATION_HOME/external directory. If you are installing the SDK on a machine that does not have the TADDM server, the JDK version 5.0 is required. If the JDK client is not on the same machine as the JDK server, then the JDK levels must match. For example, do not try to run a JDK 6.0 client with JDK 5.0 server.</td>
</tr>
</tbody>
</table>

### TADDM SDK installation

This section describes how to install the TADDM SDK software on your computer.

You can use the SDK in either of the following modes:
- Embedded mode: The SDK is installed when TADDM is installed on your system. See the topic on the embedded mode for more information.
- Standalone mode: Use this mode to install the SDK on standalone systems. See the topic on the standalone mode for more information.

On multiuser systems, like Linux, and AIX, if more than one person uses the SDK, the log files will collide on permissions. To avoid this, you can install the SDK in your home directory.

**Embedded mode**

If the TADDM server is already installed on your computer, the SDK is available as part of the distribution in the $COLLATION_HOME/sdk directory, as shown in the table below.

Table 2. Embedded mode directory structure

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>dist/</td>
<td>TADDM root directory. The $COLLATION_HOME directory is the directory where TADDM is installed plus the dist subdirectory. On operating systems such as AIX or Linux, the default location for installing TADDM is the /opt/IBM/taddm directory. Therefore, in this case, the $COLLATION_HOME directory is /opt/IBM/taddm/dist. On Windows operating systems, the default location for installing TADDM is the c:\IBM\taddm directory. Therefore, in this case, the %COLLATION_HOME% directory is c:\IBM\taddm\dist.</td>
</tr>
<tr>
<td>bin/</td>
<td>Standard TADDM distribution</td>
</tr>
<tr>
<td>deploy/</td>
<td></td>
</tr>
<tr>
<td>etc/</td>
<td></td>
</tr>
<tr>
<td>external/</td>
<td></td>
</tr>
<tr>
<td>lib/</td>
<td></td>
</tr>
<tr>
<td>log/</td>
<td></td>
</tr>
<tr>
<td>sdk/</td>
<td>Contains the TADDM SDK. See the table on the Standalone mode directory structure for more information.</td>
</tr>
</tbody>
</table>

**Standalone mode**

To install the TADDM SDK separately, go to the $COLLATION_HOME/sdk directory and extract the sdk.zip file to any directory on your system. The directory structure of the extracted SDK is shown in the following table:

Table 3. Standalone mode directory structure

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptor</td>
<td>Contains TADDM Discovery Library Adapter 1.0.</td>
</tr>
<tr>
<td>bin</td>
<td>Contains useful shell scripts and batch files</td>
</tr>
<tr>
<td>dla</td>
<td>Contains IBM Discovery Library IDML Certification Tool</td>
</tr>
</tbody>
</table>
Table 3. Standalone mode directory structure (continued)

<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>doc</td>
<td>Contains English pdfs and other documentation files</td>
</tr>
<tr>
<td>etc</td>
<td>Configuration properties</td>
</tr>
<tr>
<td>examples</td>
<td>Samples directory</td>
</tr>
<tr>
<td>lib</td>
<td>Server and client runtime libraries</td>
</tr>
<tr>
<td>log</td>
<td>Runtime logs</td>
</tr>
<tr>
<td>schema</td>
<td>The XML Schema</td>
</tr>
</tbody>
</table>

Configuring the TADDM SDK

You can configure the TADDM SDK by specifying values for environment variables. You can also optionally configure the operation of the SDK by specifying values for configuration parameters.

Setting environment variables

Before you begin

You must set environment variables before using the Command Line Interface (CLI) and software developer kit utilities, or before running the supplied examples.

Procedure

To set the environment variables, complete the following steps:

1. Set the JAVA_HOME environment variable to the directory for the JRE 5.0.
   
   If JAVA_HOME is not set, the script runs the first Java executable file found on the execution path.

2. Set the COLLATION_HOME environment variable to the TADDM root directory.

Setting configuration properties

The configuration parameters are in the $COLLATION_HOME/sdk/etc/collation.properties file. Table 4 describes the configuration parameters you can specify:

Table 4. Configuration properties

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.collation.version</td>
<td>Version of the API</td>
</tr>
<tr>
<td>com.collation.LogFile</td>
<td>Location that client side messages are logged in. The directory must exist. The file is created if it does not exist. The supplied default is ../log/api-client.log. If this property is not specified, logging defaults to stdout.</td>
</tr>
</tbody>
</table>
Table 4. Configuration properties (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.collation.log.level</td>
<td>Logging level, from among the following values:</td>
</tr>
<tr>
<td></td>
<td>• INFO—Default</td>
</tr>
<tr>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td></td>
<td>• DEBUG</td>
</tr>
<tr>
<td>com.collation.log.filesize</td>
<td>Log file size. The default value is 20 MB.</td>
</tr>
<tr>
<td>com.collation.log.filecount</td>
<td>Rollover count. The default value is 3.</td>
</tr>
<tr>
<td>com.collation.proxy.api.port</td>
<td>Default port for the TADDM SDK. This value must be the same as the setting for the TADDM server. The default value is 9530. If the API connects to multiple TADDM servers, you must configure all servers to use the same port, or specify the port when connecting.</td>
</tr>
<tr>
<td>com.collation.api.ssl.port</td>
<td>Default SSL port for the TADDM SDK. The specified value must be the same as the setting for the TADDM server. The default value is 9531. If the API connects to multiple TADDM servers using SSL, you must configure all servers to use the same SSL port, or specify the port when connecting.</td>
</tr>
</tbody>
</table>

The com.collation.proxy.api.port property setting must match the setting for the TADDM server. Otherwise the TADDM SDK does not work. This is required for both embedded and standalone modes.

Verifying the SDK installation

**Before you begin**

You can verify that you successfully installed and configured the TADDM SDK.

**Procedure**

To verify successful installation, complete the following steps:

1. Change to the SDK binary directory by running a command similar to the following:
   ```
   cd $COLLATION_HOME/sdk/bin
   ```
   **Note:** Windows users: The instructions for verification on Windows are similar, except `api.bat` is used instead of `api.sh`, and the bin directory is located in `%COLLATION_HOME%\sdk\bin`.

2. Display the CLI usage by running the following command:
   ```
   % ./api.sh
   ```

3. Display the discovery status by running the following command:
   ```
   % ./api.sh -u user -p password -H host discover status
   ```
This command queries the current discovery status. If you see a valid status (such as Idle), you have successfully communicated with the TADDM server and run a command.

4. Start a discovery by running the following command:
   
   % ./api.sh -u user -p password -H host discover start 10.10.10.12

   Then check the discovery status to verify that the discovery is running:
   
   % ./api.sh -u user -p password -H host discover status

5. Query the defined discovery scopes by running the following command:
   
   % ./api.sh -u user -p password -H host find Scope

   The command returns an XML file with the scope.

6. Collect all computer systems in the discovery by running the following command:
   
   % ./api.sh -u user -p password -H host find ComputerSystem

   The command returns an XML file containing all discovered computer systems.

Using the TADDM SDK as a software component

To integrate the TADDM SDK into an application or into an application server environment, you must set the compilation and runtime class paths, and set the access control.

Before you begin

The class paths point to the Java library that provides the Java API.

The TADDM SDK distribution also bundles the saxon and xalan libraries for XSLT and XQuery processing. You can use these libraries, or your own XML processing tools for XSLT and XQuery processing.

Procedure

To integrate the SDK as a component, complete the following steps:

1. Set the following class path at both compilation and runtime:

   CLASSPATH=$COLLATION_HOME/sdk/lib/taddm-api-client.jar:
   $COLLATION_HOME/sdk/lib/platform-model.jar

2. Configure the access settings (user ID and password).

   To use the Java and CLI API, you must configure the access settings using the Discovery Management Console. You can use the same user ID and password for API access and the Discovery Management Console.

What to do next

After upgrading from a previous TADDM release, you might need to update the class path to include the correct .jar files.

The .jar files in the $COLLATION_HOME/sdk/lib directory are also used by the TADDM server. Therefore, the SDK file should not be moved after installation. If you need to have the SDK files in a different location, you can extract them from the sdk.zip file on the product DVD.
Required Java .jar files

The taddm-api-client.jar and platform-model.jar files are required to use the Java API and must be present in a directory listed on the system CLASSPATH environment variable. These files are provided in the lib subdirectory of the SDK directory.

The taddm-api-client.jar and platform-model.jar files have replaced all previous TADDM JAR files as the archives that contain client APIs and model definitions.

If you are using the IBM Tivoli Business Service Manager (TBSM) XML toolkit with the JDBC connection type, you also need oal-topomgr.jar. You can download this JAR file from the following location:

http://taddm.server.machine.name:9430/GetTaddmVersion/getVersion/getoaltopomgrfile

To detect changes to the version of the JAR files on the TADDM server, a client application can use the following URLs to obtain checksum values for the files:

- taddm-api-client.jar:
  http://taddm.server.machine.name:9430/GetTaddmVersion/getVersion/clientjar

- platform-model.jar:

- oal-topomgr.jar:

where taddm.server.machine.name is the fully qualified domain name of the server where TADDM is running.

Note: If the TADDM server is started as part of the installation process, the checksum for taddm-api-client.jar is reported incorrectly as 11111111 afterward. If this happens, restart the server; subsequent client requests return the correct checksum.

A client can also check the version of the TADDM server by using the following URL:


This URL returns the product version (for example, 7.2.1).

SOAP API installation and configuration

The SOAP API is installed with the TADDM SDK. However, you need to complete the procedure described in this section before using the API.

Procedure

To complete the SOAP API installation and configuration, complete the following steps:

1. Download the Axis package from the Internet.
2. Uncompress the package to the $COLLATION_HOME/sdk/lib directory.
3. Include the JAR files in the Axis package in the CLASSPATH.

REST API installation and configuration

The TADDM REST API does not require any .jar files supplied with TADDM; however, some .jar files might be required if you want to work with TADDM model objects.
About this task

You can use the TADDM .jar files to access the TADDM model object classes and the ModelObjectFactory class, which converts model objects to and from XML representations.

Procedure

To access the classes, complete the following task:

Include the appropriate .jar files for your JDK version in a directory on your CLASSPATH environment variable:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK 5.0</td>
<td>$COLLATION_HOME/sdk/lib/taddm-api-client.jar</td>
</tr>
<tr>
<td>JDK 5.0</td>
<td>$COLLATION_HOME/sdk/lib/platform-model.jar</td>
</tr>
</tbody>
</table>

Understanding the Common Data Model

The Common Data Model (CDM) is the definitional language used to integrate understanding and the exchange of data between Tivoli management products concerning resources and components of a customer's business. The CDM is the model used to communicate details about resource instances with the IBM Tivoli Application Dependency Discovery Manager (TADDM) database.

The CDM is entirely composed of data definitions. These definitions are characteristics that identify resources, their meanings, and any restrictions on their lengths or values. The content of the CDM is obtained by the merging of applicable industry information and data model standards and the data models used by our current products into a single, converged model. It incorporates the following standards:

- Distributed Management Task Force (DMTF) Common Information Model (CIM) standard
- The following Business Process standards:
  - Business Process Execution Language (BPEL),
  - IT Infrastructure Library (ITIL) specification
  - LDAP directory schema
- The following domain specific standards:
  - TeleManagement Forum (TMf),
  - Storage Networking Industry Association (SNIA), and more.

The Common Data Model is in use by multiple applications, including TADDM. The applications that use the CDM are able to share definitions and terminology for resource instance data that is common between them, enabling the construction of higher-level applications that encompass the overall management environment and share information between those systems. The CDM describes the input and output contents of the TADDM API, sensors, utility applications, and Discovery Management Console.

The CDM is different from a schema. A schema, is usually associated with a database, includes both the organization of data into a logical model and the
specification of how that data is stored in specific columns of specific tables (also known as the physical model of the database). The CDM represents a logical model composed of definitions that enables consistent identification of resource instances, information about them, and relationships between them. The data model links business and IT processes with the systems that provide them, the users that invoke them, the policies that control them, the resources that processes use, and much more. The CDM classifies and organizes the most commonly managed characteristics of users, resources, and business IT information and processes and presents them in a way that all applications can use.

For more details on CDM, see the following information:

- Tivoli Common Data Model Web site in the $COLLATION_HOME/sdk/doc/model directory

The Common Data Model has the following characteristics:

- It does not define the physical schema, nor does it define how a management system operates.
- It defines the resources and characteristics of a management environment that the management system monitors, analyzes, and controls.
- It is also in use when management applications exchange information about resource instances and their relationships to other resources.
- It standardizes the characteristics, the concepts of classes, attributes, interfaces, naming rules, naming policies and the data types that are in use.
- It provides consistent definitions of items, best practices for content, and guidelines for mapping resource instance data to the CDM.

The Common Data Model includes the following objects:

Classes

Have the following characteristics or rules:

- A Class is a construct used to group related attributes.
- Classes are the representation of a resource instance type (for example, an OperatingSystem as a type of resource instance).
- As the basic structure of the model, classes contain attributes, implement interfaces, and can optionally be involved in relationships.
- Classes are hierarchical and inherit the properties of parent classes.
- Classes can also explicitly include properties that pertain to a level of detail.
- Instances of classes represent the actual resource instances, the *nouns* representing the physical or logical resources in the environment.
- Instances have attributes and can take part in relationships. For example, in a database management environment, items such as the database server, tables, and connections are Instances.

**Note:** Instances also include things that are not limited to being managed but which take part in the management process, such as users or business systems.

- Out of the various objects in the CDM, Classes are the only ones in use to represent resource instances. There are particular classes mentioned throughout the TADDM documentation that have particular meaning:
- **ModelObject** - This class represents the base or root class in the CDM. All classes derive in some way from ModelObject. The term ModelObject is used in the documentation to represent any defined class in the CDM.

- **ManagedElement** - This is another representation of a base or root class in the CDM, and directly corresponds to the DMTF Common Information Model representation with the same name. The term ManagedElement is also used in the documentation to represent any defined class in the CDM. The ModelObject and ManagedElement classes are used interchangeably.

- **ManagementSoftwareSystem** - Also known as a MSS, this class represents the management products that are providing data to TADDM through some mechanism. Each provider of data (including TADDM’s sensors) are represented as a resource instance of the type ManagementSoftwareSystem.

The CDM supports specialization through single inheritance, although the use of interfaces gives the model some aspects of multiple inheritance. All classes are organized into a single-rooted, single inheritance hierarchy with the **ModelObject** class as the root. Every class, with exception of ModelObject, specifies exactly one parent, and the child class inherits all characteristics of the parent class.

The CDM additionally includes naming rules for model objects that specify the attributes required to uniquely name objects in the CMDB. See the section on **Naming instances** for more information about naming rules for model objects.

- **Persistent vs. Non-Persistent classes:**
  - A persistent class is a class whose instances can be stored in a database, whereas instances of a non-persistent class cannot be stored in a database.
  - When using MQL (Model Query Language), you can only query objects of persistent classes. The only exception is when you query the attribute, “guid” of a ModelObject (non-persistent class), as in the following example:
    - The attribute, “members”, is a ModelObject, and the following queries will return the same results:
      ```sql
      SELECT * 
      FROM AccessCollection 
      WHERE members.guid = "E72B13789C9039BFB32E3822FE50C197"
      
      SELECT * 
      FROM AccessCollection 
      WHERE members == "E72B13789C9039BFB32E3822FE50C197"
      ```
  - In the model Javadoc (Javadoc for TADDM’s CommonDataModel), if the tag, **Persistable**, is set to true for a given class, then it is a persistent class. If the tag is not present for a given class, then it is a non-persistent class.
    - Examples of persistent classes: ComputerSystem, SoftwareModule, AppServer
    - Examples of non-persistent classes: ModelObject, Database, LogicalElement

**Attributes**

Have the following characteristics or rules:

- An attribute defines a particular property that is valid for a class.
Each attribute has a particular meaning or semantic in terms of expected content.

Attributes are specified on CDM classes as well as interfaces.

Instances of attributes are the adjectives that describe characteristics of instances and serve to differentiate instances of the same class, such as the different Manufacturer of instances of the class ComputerSystem.

When a resource instance is created, there is the ability to store data for any attribute valid for a resource instance.

Not all attributes are required to contain a value, however, there are some attributes that are in use to represent a unique identity for a resource instance. These attributes are often referred to as identity attributes.

Interfaces

Enable the convenient reuse of a set of attributes and provide increased flexibility in the definition of relationships. For example, the attribute VersionString is a valid attribute for several different (class) types of resource instances. Rather than duplicating the attribute across multiple classes in the CDM, an interface is created to represent the set of attributes that pertain to version data.

Resource instances cannot be based on an interface. Any class that implements an interface automatically receives the set of attributes and relationships from the interface as if they existed on the class. Interfaces are hierarchical and can derive their attributes and relationships in the parent interface from inheritance.

There is a particular interface mentioned throughout the TADDM documentation that has a particular meaning. This interface is called a Configuration Item. The interface Configuration Item is used to denote particular classes in the CDM of which instances act as a Configuration Item defined by the corresponding ITIL term. Certain classes in the CDM, such as financial data, are not defined to be Configuration Items, as the CDM represents aspects from various environments.

Relationships

Have the following characteristics or rules:

- Associations between two resource instances, showing how resource instances are related to each other.
- Relationships can only be between classes, and are between classes of the same or different types.
- Each relationship has a particular definition, or type. These different relationship types carry a certain semantic that pertains to the kind of association between the resource instances.

For example, one of the relationship types in the CDM is manages, which represents the source instance participates in a controlling role to the target resource instance in the relationship. Another relationship type is installedOn, which represents the source instance as an object that is installed on the target resource instance. Both of these relationships can be valid on resource instances where the source is a instance of the class Agent and the target is a instance of the class OperatingSystem, however the two relationships have very different meanings. There can be multiple relationships between the same two classes (and the same two resource instances). Each relationship forms an association between two instances.
In the CDM each relationship instance has a source and a target, which are the relationship's roles. The number of instances that can take part in each role is important. Certain relationships only allow one instance to take part. Others allow any number of instances. The number of instances that can participate in each role is known as the cardinality of the relationship.

**Data Types**

The information contained in attributes and measurements must be presented in a well-known syntax, and for this purpose the CDM defines a set of data types that should be used for representing entity information.

The data types defined in the model do not specify a physical representation for the data, rather they specify the lengths of data and sometimes the encoding or best practice for the content of the data.

The model also includes enumerated data types that enable products to understand the common meaning of certain values.

**Naming instances**

Names (or naming attributes) form the basis for identification of resources and reconciliation between resource instances that represent the same object in the data center.

Naming is based on the generation, use, and sharing of human-readable attributes for identifying resource instances. By grouping the content of particular attributes together for a resource instance, a unique name is created for the resource instance. Given the size of the data model, there are many potential ways to name a resource instance. In order to organize the method of generating a unique name, the Common Data Model uses the concept of naming rules to group a set of attributes together that constitutes a unique identity.

**Naming rules**

A naming rule is a specification of how to name instances of a particular class, such as resources, people, and systems.

Naming rules contain a set of attributes that are required in order to name a given resource. The usual case for a naming rule is to group attributes together in order to form unique identity. If the name of two instances is the same, the instances are assumed to refer to the same entity. For example, different entities in a Layer 2 network are commonly identified in the same way, using a MAC address, even though the entities are instances of different, possibly unrelated classes. MAC addresses, by their structure, form a space from which all valid names for a station on a Layer-2 network can be assigned.

Note: This is separate from the type of network that is involved, which could be 10-BaseT, 1000-BaseT, or Token Ring.

There are two special cases where naming rules will contain more than just attributes.

1. Naming Context:

   Sometimes in the naming of a resource instance, there is a minimal amount of information available to uniquely name the instance based on the attributes that are available on the class. In cases such as these, certain naming rules specify a relationship in addition to a set of attributes, as required for the
naming rule. These relationships place what is known as a Naming Context on
the resource instance, and require a second resource to be in use to contextually
identify another resource instance.

For example:
• All that is known about a particular instance of an Operating System is the
type of Operating System.
• The attribute representing the type of a Operating System is not unique
even enough to create a unique resource instance representing the Operating
System.
• In order to use this attribute, the naming rule specifies a required
installedOn relationship from the instance of the Operating System to a
instance of a ComputerSystem (there is a implied requirement to also create
a valid instance of a Computer System in order to create the relationship).

2. NOT:
Certain naming rules are in place with a defined set of attributes that are
acceptable to uniquely name a resource instance in a majority of circumstances.
However, there are cases in the Common Data Model where another naming
rule is needed to further refine the identity of a resource, using the same set of
attributes in use by another naming rule while adding additional attributes.
Because the method to create a unique instance is based on satisfying naming
rules, it is not desirable to have a naming rule with less specific requirements
to generate a identity when more specific attributes are provided. In order to
prevent the less specific naming rule from being used, certain naming rules use
an OmittedIdentifier statement on a attribute. This is also referred to as “NOT”
in the Common Data Model Web site section on naming rules.

**Note:** You can find the Common Data Model Web site in the
$COLLATION_HOME/sdk/doc/model directory.

When this NOT operation is mentioned, the operation shows that the attribute
must be null. If any content exists in the attribute mentioned in the
OmittedIdentifier (NOT) operation, the naming rule is not used to uniquely
identify a resource. For example:
• A naming rule exists on the class Activity called ActivityName.
• This naming rule requires the attribute ActivityName to contain a value.
  – The assumption with this particular naming rule is the name of the
    activity is globally unique within the customer environment.
• In the circumstances where Activity names are not unique, there is a second
  naming rule, called QualifiedActivity.
  – This rule requires the attribute ActivityName and an owns relationship
    from a instance of the class OrganizationalEntity to the instance of the
    class Activity
• Because the naming rules use a common attribute, ActivityName, and one
  naming rule is a further refinement of another naming rule, only one naming
  rule should be used to name the instance of Activity.
• Therefore, the naming rule ActivityName specified the NOT operation on the
  owns relationship. This means that the owns relationship must not be
  populated in order to use the ActivityName naming rule.

Identification is based on the generation, use, and sharing of a machine-readable,
concise, and unique value for the purpose of processing identification. Resource
instances that are represented by the Common Data Model have both names and
identifiers:
• The names are longer, mainly alphabetic strings that people use to refer to the entities.
• Identifiers are shorter, dense, mainly numeric values that the management system uses to uniquely identify the entities.

**TADDM Globally Unique Identifiers**

Identification values are commonly referred to as globally unique identifiers (GUIDs). The TADDM GUIDs are built according to UUID version 3 specification (IETF Standards Track RFC 4122), and are used as identifiers of configuration items (CIs).

Version 3 GUIDs are generated by processing a string with an MD5-type cryptographic algorithm. TADDM passes a string that is constructed from the values of the attributes that are used in the naming rules to the GUID generation component. Most CIs have multiple naming rules and can therefore generate multiple GUIDs. The attribute values that are available when the CI is created determine which GUIDs are generated. Generally, the first GUID that is generated for an object is considered the master GUID or primary identifier for that object. Other generated GUIDs are aliases of the master GUID.

If the CIs are discovered with the same attributes and values, they always have the same set of GUIDs. However, the first GUID, which later becomes a master GUID, is generated randomly. That is why a particular CI might not have the same master GUID on different TADDM installations. Likewise, it might not be chosen again when the item is deleted or the database is re-created. The same types of CIs, such as ComputerSystems, might also use GUIDs that are calculated from a different naming rule than their master GUIDs.

Generally, TADDM application programming interfaces (APIs) identify CIs by their master GUIDs, but they can also identify them by their aliases. That is why, if you want to find a particular CI, you can search for it by using its alias GUID.

**GUID erosion**

GUIDs that are aliases of a master GUID might erode during the lifecycle of a configuration item. Erosion happens when an attribute that defines a single naming rule, such as a signature, changes. After this change, a new set of GUIDs is generated, and replaces the old values. If the attributes of a master GUID change, this GUID remains the same and a new alias is added.

**Master GUID changes**

A master GUID of a particular configuration item can change due to any of the following conditions:

**Deletion of a configuration item, and rediscovery**

When a configuration item is deleted from the TADDM database, a different GUID might be chosen as a master GUID during the next store of this CI.

**Configuration items merge scenario**

When new data is available in TADDM, two different CIs might be identified as the same instance. A user can also start the merge manually. In this scenario, the attributes of a transient and a durable CI can merge. As a result, the master GUID of a transient CI becomes a new alias of the durable one, and the master GUID of the durable CI represents a CI that was created after the merge.
TADDM upgrade
When you upgrade to a new version of TADDM, the attributes that are part of naming rules might change. This situation might also affect the data migration process that is supposed to ensure that master GUIDs remain the same after the upgrade. A new version of sensors or Discovery Library Adapters might also change the way the attribute values are stored.

Class names
The TADDM Common Data Model class object names can be referenced by either their long name or their short name. Most object names can be referenced by their short name.

For a computer system, the short name is ComputerSystem and the long name is com.collation.platform.model.topology.sys.ComputerSystem.

The exception to the usage of short names is in the case of duplicates. For example, SSLSettings must be referenced by its long name because there are 2 instances of SSLSettings:
• com.collation.platform.model.topology.app.lotus.SSLSettings
• com.collation.platform.model.topology.app.SSLSettings.

The following code sample displays all the short and long names for classes in the Common Data Model. Once you run this command, the duplicate class names which must be referenced by their long name are listed at the end of the results.

```
DisplayClassNames sample
import com.collation.proxy.api.client.*;
import com.collation.proxy.api.util.*;
import com.ibm.cdb.api.ApiFactory;
import java.util.*;

class DisplayClassNames {
    public static void main(String[] args) {
        CMDBApi api = null;
        try {
            System.out.println("--- Displaying Model Object Names ----");
            ApiConnection conn = ApiFactory.getInstance().
                getApiConnection("localhost", -1, null, false);
            ApiSession sess1 = ApiFactory.getInstance().getSession(conn,
                "administrator", "collation", ApiSession.DEFAULT_VERSION);
            api = sess1.createCMDBApi();
            String[] classNameArray = api.getClassNames();
            ArrayList shortNames = new ArrayList(classNameArray.length);
            ArrayList dups = new ArrayList(10);
            for (int i = 0; i < classNameArray.length; i++) {
                // print the short and long class names
                System.out.println("Short Name = " + classNameArray[i]);
                System.out.println("Long Name = " + classNameArray[i+1]);
                // See if short name is a dup
                if (shortNames.contains(classNameArray[i])) {
                    dups.add(classNameArray[i]);
                } else {
                    shortNames.add(classNameArray[i]);
                }
            }
        }
    }
}
```
Dependencies between resources

TADDM discovers and categorizes several types of cross-tier dependencies, these dependencies are reflected in the CDM. Dependencies model the runtime relationships among the various components within the CDM.

There are several types of dependencies, including:

- Transactional dependencies
  Transactional dependencies occur between application components, such as web servers, application servers, and databases. The dependent component issues requests to the provider component in order to perform certain functions. For example, a Java Database Connectivity (JDBC) connection from a Java 2 Platform, Enterprise Edition (J2EE) server to a database is a transactional dependency. In this case, the provider is often called a server and the dependent called a consumer or client.

- Service dependencies
  Service dependencies occur between application components and infrastructure services, such as Domain Name System (DNS), Lightweight Directory Access Protocol (LDAP), and Network File System (NFS). The provider is the infrastructure service, and the dependent component requests system services from the provider. For example, a request to map a DNS name to an IP address.

- IP dependencies
  IP dependencies occur between two computer systems or between an application server and a computer system. TADDM creates this type of relationship when it discovers a relationship between two computer systems but cannot discover exactly which application server is involved.

- System dependencies
  System dependencies occur between an application server and its host computer system.

- Application to application dependencies
  Application to application dependencies occur from one business application to another business application.

Example of dependencies

When transaction dependencies are created for two application servers, then IP dependencies are not created between them. Neither are IP dependencies created between the application server and their hosts. However, there can exist another
logical connection for example between two processes and based on these connections IP dependencies can be created between computer systems. For example, consider the following scenario:

- Computer system (CS1) hosts an application server (AP1) and process (P1)
- Computer system (CS2) hosts an application server (AP2) and process (P2)

There are two logical connections created by TADDM: AP1 <-> AP2 and P1<->P2

In this scenario, a transactional dependence is created between AP1 and AP2 (based on the logical connection AP1 <-> AP2). An IP dependency is created between CS1 and CS2 (based on the logical connection between P1<->P2).

Changes to the Common Data Model

This section lists the changes to the Common Data Model.

For more details, refer to the Common Data Model documentation in the $COLLATION_HOME/sdk/doc/model/CDMWebsite.zip file of your TADDM installation DVD.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM High-Availability Cluster Multi-Processing (HACMP™)</td>
<td>New classes have been added to represent the IBM HACMP cluster, nodes, and resources.</td>
</tr>
<tr>
<td>Sybase database</td>
<td>New classes have been added to represent the Sybase remote server, users, logins, roles, tables, table partitions, thresholds, and time ranges.</td>
</tr>
<tr>
<td>Oracle Automatic Storage Management (ASM)</td>
<td>New classes have been added to represent the Oracle ASM.</td>
</tr>
<tr>
<td>Oracle Real Application Cluster (RAC)</td>
<td>New classes have been added to represent the Oracle RAC.</td>
</tr>
<tr>
<td>VMware vSphere</td>
<td>New classes have been added to represent VMware vSphere.</td>
</tr>
<tr>
<td>File systems</td>
<td>New attributes have been added for NFS and UNIX file systems.</td>
</tr>
<tr>
<td>Operating system</td>
<td>A new attribute has been added to represent the number of bits per word for the operating system.</td>
</tr>
<tr>
<td>Computer system</td>
<td>New attributes have been added to represent the BIOS and the L2 cache size.</td>
</tr>
<tr>
<td>Network</td>
<td>A new attribute has been added to represent the IP address of the TCP/IP gateway configured for the network interface.</td>
</tr>
<tr>
<td>Software</td>
<td>A new attribute has been added to represent the word size.</td>
</tr>
</tbody>
</table>

TADDM API overview

All discovery data that is displayed using the Discovery Management Console is accessible using the TADDM Application Programming Interfaces (API). This topic describes the principal TADDM APIs: the Java API, the SOAP API, the REST API, and the Command Line Interface API.
**Application programming interface overview**

You can access topological elements of TADDM through the find() methods of the application programming interface (API.)

To access topological elements of TADDM through the find() methods of the API, using any of the following methods:

**Java API**

The complete TADDM API, enabling Java application development and integration.

**SOAP API**

Exposes elements of the TADDM API as a Web service.

**REST API**

Exposes elements of the TADDM API as a RESTful Web service.

**Command-line interface API**

Provides a wrapper around the Java API to enable access from the command line for scripting, simple customizing, and scheduling.

**XML schema overview**

The TADDM XML schema flattens the hierarchical structure of the Common Data Model into an XML document, with most contained objects embedded within the document.

TADDM API methods return an XML document containing a list of objects specified by the Model Query Language (MQL) query, where applicable. This document is larger than the original data but is easier to search using tools such as XQuery or XPath.

The TADDM SDK represents the dependencies between objects using independent dependency objects, which connect providers with dependent services using object IDs.

When formatting XML documents for use with the TADDM SDK, keep in mind the following considerations:

- XML is a hierarchical model and does not permit cycles.
- The property_nameX is a ModelObject.
- The abbreviated_searched_class_name is the searched class name and not the actual class name.
- The xsi:type and GUID are XML attributes, and are not represented as separate elements.
- Array: When the element is part of an array, N is its index.

The following table describes the XML document structure:

<table>
<thead>
<tr>
<th>XML</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</td>
<td>Header</td>
</tr>
</tbody>
</table>
### Table 6. XML document structure (continued)

<table>
<thead>
<tr>
<th>XML</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;results xmlns=&quot;urn:www-collation-com:1.0&quot; xmlns:coll=&quot;urn:www-collation-com:1.0&quot; xmlns:xsi=&quot;http://www.w3.org/2001/XMLSchemaInstance&quot; xsi:schemaLocation= &quot;urn:www-collation-com:1.0 urn:www-collation-com:1.0/results.xsd&quot;&gt;</td>
<td>Top-level result node, including the XML namespace specification</td>
</tr>
<tr>
<td>&lt;abbreviated_searched_class_name xsi:type=&quot;full class name&quot; array=&quot;N&quot; LINK=&quot;guid&quot; lastModified=&quot;Last Modified Time in ms&quot; guid=&quot;unique model object id&quot;&gt;</td>
<td>Class attributes, such as last modified time and unique object ID</td>
</tr>
<tr>
<td>&lt;property_name1&gt;&quot;text value&quot; &lt;/property_name1&gt; ...</td>
<td>Attribute values and embedded objects</td>
</tr>
<tr>
<td>&lt;property_nameX xsi:type=&quot;full class name&quot; guid=&quot;unique model object id&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;property_nameN&gt;&quot;text value&quot; &lt;/property_nameN&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/abbreviated_searched_class_name&gt;</td>
<td>End of values</td>
</tr>
<tr>
<td>&lt;/results&gt;</td>
<td>End of results</td>
</tr>
</tbody>
</table>

### JSON format overview

The REST API uses JSON format to return data representing model objects; you can request JSON output by specifying the `feed=json` parameter on a REST API call.

The following table describes the structure of the JSON format used to represent model objects.

<table>
<thead>
<tr>
<th>JSON element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[</td>
<td>Begins an array of model objects.</td>
</tr>
<tr>
<td>{</td>
<td>Begins a model object, which might contain zero or more model objects.</td>
</tr>
<tr>
<td><em>name</em>:value,<em>name</em>:value</td>
<td>One or more name-value pairs, separated by commas.</td>
</tr>
</tbody>
</table>
JSON element | Description
---|---
"_class": "class_name" | A required name-value pair containing the name of the model object. The model object class name can be in either of two forms: • Short name (for example, ComputerSystem) • Fully qualified name (for example, com.collation.platform.model.topology.sys.ComputerSystem)

If you specify longClassName=true on a REST query, then all of the returned values for _class contain the full model name. Otherwise, the short name is returned unless it is not unique (in which case the full name is returned).

}] | Ends a model object.

}] | Ends an array of model objects.

The following example shows JSON output from a ComputerSystem query at a depth of 1:

```json
[
  {
    "displayName": "esx3-vm16-rhes4",
    "devices": ["_class": "DiskDrive", "guid": "2A282766BE0B3465A955DE54BD3F6AB5"],
    "L2Interfaces": ["_class": "L2Interface", "guid": "FA04699199A9538A5A958B0496017A776"],
    "createdBy": "system",
    "class": "LinuxUnitaryComputerSystem",
    "controllers": ["_class": "Controller", "guid": "7B72D3B5448D30388F9D9497EA8F971A82"],
    "CPUSpeed": 3191000000,
    "fqdn": "esx3-vm16-rhes4",
    "memorySize": 398873104,
    "ipInterfaces": ["_class": "IpInterface", "guid": "6300742848BA3978EAE4F47090DF7A"],
    "signature": "9.43.73.87(000C29A467A9)",
    "systemId": "2b095749",
    "bidiFlag": 3,
    "name": "esx3-vm16-rhes4",
    "OSRunning": ["_class": "Linux", "guid": "04BFBCD2A173325BF8C95CD2BD191AF"],
    "CPUType": "Intel(R) Xeon(TM) MV",
    "type": "ComputerSystem",
    "numCPUs": 1,
    "architecture": "i686",
    "fileSystems": ["_class": "UnixFileSystem", "guid": "CDA94FFBB848300AAB2A421E6EE6234"],
    "lastModifiedTime": "122600427541"
  }
]
```

**Model Query Language overview**

The `find()` command from the TADDM API accepts a query string, specified using the Model Query Language (MQL). The MQL acts as a filter to limit the selected objects.

MQL uses a SQL-like syntax to specify the model object class or other data sources, their attributes, along with a filter expression.

The syntax of an MQL query is as follows:

```sql
SELECT attribute-list FROM data-sources [ WHERE expression ]
```
Table 7 describes the elements of an MQL query. MQL is not case-sensitive.

Table 7. MQL query elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute-list</td>
<td>The value *, or a comma-separated list of attributes of the source ModelObject class. Embedded attributes can also be specified. An attribute name always starts with a lowercase letter, unless the first and second letter are both uppercase. Subsequent letters in the attribute name can be uppercase or lowercase. Examples of attribute names include the following: · displayName · fqdn · OSRunning</td>
</tr>
<tr>
<td>data-sources</td>
<td>A comma-separated list of model object class names. These classes must be persistent, since you cannot query non-persistent objects.</td>
</tr>
<tr>
<td>expression</td>
<td>The filter expression, expressed using the following format: member-name OP expression [ ... ] where: · member-name is an attribute of the selected data source, and can include dot separated members (The member classes specified in the query expression must be persistent. You can query the member attributes to get the values that match the query expression.) · OP is an operator · expression is a statement that returns a value For example: SELECT * FROM ComputerSystem WHERE OSRunning.OSName == 'Linux' In this case, OSRunning is an OperatingSystem object referenced by a ComputerSystem (which is a persistent object), and OSName is a primitive member. See Table 8 for a description of operators and associated precedence.</td>
</tr>
</tbody>
</table>

Table 8 describes the MQL operator precedence, with higher values representing greater precedence.

Table 8. MQL Operator Precedence

<table>
<thead>
<tr>
<th>Token</th>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td>logical OR</td>
<td>1</td>
</tr>
<tr>
<td>and</td>
<td>logical AND</td>
<td>2</td>
</tr>
<tr>
<td>instanceof</td>
<td>is instance of</td>
<td>3</td>
</tr>
<tr>
<td>==</td>
<td>equals</td>
<td>3</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>3</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>3</td>
</tr>
<tr>
<td>Token</td>
<td>Operator</td>
<td>Precedence</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td>------------</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal</td>
<td>3</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>3</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal</td>
<td>3</td>
</tr>
<tr>
<td>starts-with</td>
<td>starts-with</td>
<td>4</td>
</tr>
<tr>
<td>ends-with</td>
<td>ends-with</td>
<td>4</td>
</tr>
<tr>
<td>equals</td>
<td>equals</td>
<td>4</td>
</tr>
<tr>
<td>not-equals</td>
<td>not equals</td>
<td>4</td>
</tr>
<tr>
<td>is-null</td>
<td>is null</td>
<td>4</td>
</tr>
<tr>
<td>is-not-null</td>
<td>is not null</td>
<td>4</td>
</tr>
<tr>
<td>in</td>
<td>in</td>
<td>5</td>
</tr>
<tr>
<td>()</td>
<td>parentheses</td>
<td>5</td>
</tr>
<tr>
<td>exists</td>
<td>array contains</td>
<td>5</td>
</tr>
<tr>
<td>upper()</td>
<td>function</td>
<td>5</td>
</tr>
<tr>
<td>lower()</td>
<td>function</td>
<td>5</td>
</tr>
<tr>
<td>!</td>
<td>unary not</td>
<td>5</td>
</tr>
<tr>
<td>.</td>
<td>dot selection</td>
<td>6</td>
</tr>
<tr>
<td>contains</td>
<td>contains</td>
<td>3</td>
</tr>
</tbody>
</table>

MQL does not support the following SQL SELECT operators or features:
- GROUP BY
- HAVING
- DISTINCT
- nested SELECTs
- BETWEEN
- Aggregates

You can specify the logical operator AND as and, AND, or &&, and the logical operator OR as or, OR, or ||. In addition, you must enclose all strings using single quotation marks, for example, 'IBM'.

**Joins**

MQL supports left inner joins against model objects, as illustrated by the following example:

```sql
SELECT Db2Server.* FROM Db2Server, OracleInstance WHERE Db2Server.port == OracleInstance.port
```

This join returns all Db2Server model objects in cases when the port number of Db2Server and the OracleInstance are equal. MQL does not support combinations of right outer, left outer, full, or cross joins.

**Limitations**

On DB2 version 9.5, the equals and not-equals operators fail when they are run on attributes of the CLOB data type in the database. The following exception is thrown:

```java
com.ibm.db2.jcc.am.SqlSyntaxErrorException: DB2 SQL Error: SQLCODE=-418, SQLSTATE=42610
```
SELECT statement grammar

The following example shows the grammar of the SELECT statement. See the Javadoc for more details and the latest updates.

\[
\begin{align*}
\text{statement} & := \text{SELECT} \text{ attribute-list} \ [\text{EXCLUDING attribute-list}] \\
& \quad \text{FROM} \ [\text{ONLY}] \ \text{class-list} \ [\text{WHERE} \ [\text{expression} | \ \exists \text{expr}] ] \\
& \quad \ [\text{FETCH FIRST} \ n \ \{ \text{ROW} | \text{ROWs} \} \ [\text{ONLY}]] \ [\text{ORDER BY} \ \text{order-list}]
\end{align*}
\]

\[
\begin{align*}
\text{attribute-list} & := \text{attrib} \ {,} \ \text{attrib}* | *
\end{align*}
\]

\[
\begin{align*}
\text{class-list} & := \text{domain-class} \ {,} \ \text{domain-class}* \\
\text{class} & := \langle \text{a model object class} \rangle
\end{align*}
\]

\[
\begin{align*}
\exists \text{expr} & := \exists( \text{array-attrib op value} \ \{ \text{logical_op array-attrib op value}* \} ) \\
\text{expression} & := \{ \text{attrib op value} | \text{attrib post-op} | \text{pre-op (attrib)} \\
& \quad | [\text{NOT}] \ \text{IN} \ \{ \text{expression} \ [\ldots] \} \ \{\text{logical_op expression}\} \} \\
\text{value} & := \langle \text{data value} \rangle \\
\text{in_expr} & := [\text{NOT}] \ \text{IN} \ \{ \text{expression} \ [\text{expression} \ldots] \}
\end{align*}
\]

\[
\begin{align*}
\text{array-attrib} & := \langle \text{series of attributes where at least the second to last attribute is an array} \rangle \\
\text{op} & := != | == | > | < | >= | <= | contains | starts-with | ends-with | \\
& \quad | equals | not-equals | instanceof \\
\text{logical_op} & := \text{AND} | \text{OR} | \& \& | || \\
\text{post_op} & := \text{is-null} | \text{is-not-null} \\
\text{pre_op} & := \text{lower} | \text{upper}
\end{align*}
\]

\[
\begin{align*}
\text{attrib} & := \langle \text{class} . \rangle \ [\langle \text{an attribute of a class}\rangle . \text{embedded_attribute} | * ] \\
\text{embedded_attribute} & := \langle \text{embedded attribute}\rangle . \text{embedded_attribute} | * \\
\text{domain-class} & := \langle \text{domain-list} \rangle \ \text{class} \\
\text{domain-list} & := \langle \text{domain} \ [\text{domain}]* \rangle \\
\text{domain} & := \langle \text{the server from which to pull data from, default: local database} \rangle \\
\text{order-list} & := \text{attrib} \ [\text{ASC} | \text{ASCENDING} | \text{DESC} | \text{DESCENDING} \ [\ , \ \text{order-list} \]}
\end{align*}
\]

Attributes can contain wildcard characters. Also, all keywords, such as SELECT, FROM, and WHERE are not case-sensitive.

Examples

The following example shows an MQL query that filters for computer systems running the Linux operating system:

```
SELECT *
FROM ComputerSystem
WHERE OSRunning.OSName == 'Linux'
```

The following query uses the EXISTS operator to query array membership, matching all computer systems that have an interface listening on ibm.com or their netmask set to 255.255.255.0:

```
SELECT *
FROM ComputerSystem
WHERE EXISTS (ipInterfaces.ipNetwork.name ends-with '.ibm.com' OR ipInterfaces.ipNetwork.netmask == '255.255.255.0')
```

The following query selects all computer systems that have the attribute virtual set to true:

```
SELECT *
FROM ComputerSystem
WHERE virtual
```

The following query selects all computer systems that have the attribute virtual set to false:

```
SELECT *
FROM ComputerSystem
WHERE not virtual
```
The following query selects all operating systems that have the installed service attribute with the name that contains "Wireless". Since the installed service attribute is available only on the Windows operating system, you must use join.

```sql
SELECT OSInstalled
FROM ComputerSystem,WindowsOperatingSystem
WHERE ComputerSystem.guid==WindowsOperatingSystem.parent.guid
AND
EXISTS(WindowsOperatingSystem.installedServices.displayName contains 'Wireless')
```

The following query selects all AppServers with the primarySAP attribute that has a port specified as its value:

```sql
SELECT primarySAP.portNumber,displayName
FROM AppServer
WHERE primarySAP.portNumber==9084
```

The following query selects all RuntimeProcesses that among their ports have port 1415. The query must use the EXISTS operator because the ports attribute for RuntimeProcess is an array attribute.

```sql
SELECT ports.portNumber,displayName
FROM RuntimeProcess
WHERE EXISTS (ports.portNumber==1415)
```

**MQL queries with NOT EQUAL TO operator (!=)**

The queries with NOT EQUAL TO operator do not return results that contain an attribute that is set to NULL because such a result is evaluated to "unknown".

**Example**

It is often assumed that the number of results that are returned from the following API find command:

```bash
./api.sh -u <admin> -p <pass> find --count "select * from ComputerSystem"
```

equals the sum of the results that are returned from the following two API find commands:

```bash
./api.sh -u <admin> -p <pass> find --count "select * from ComputerSystem where manufacturer == 'IBM'"
./api.sh -u <admin> -p <pass> find --count "select * from ComputerSystem where manufacturer != 'IBM'"
```

However, the manufacturer attribute is set to NULL, therefore it is excluded from the results that are returned from the query that contains NOT EQUAL TO operator.

The queries that contain the NOT EQUAL TO operator can have the following forms:

```sql
select * from ComputerSystem where manufacturer != 'IBM'
select * from ComputerSystem where not(manufacturer == 'IBM')
```

If you want to select all ComputerSystems with manufacturer other than IBM, use the following query:

```sql
select * from ComputerSystem where manufacturer != 'IBM'
or manufacturer is-null
```

**Using the Java API**

The Java API provides control over the discovery process and aspects of the Common Data Model including access to the resulting model data.
Using the Java API, you can create applications that add, update, and delete model objects. You can query model objects by class name or object ID number. You can also use the interface to manage relationships between objects, perform comparisons, and examine the change history.

The model data can be filtered on the server and is returned to the client in XML format. You can then perform transformations and further queries on the client, as required. The Java API also offers methods you can use to manage sessions and implement security-related operations.

The Java API is contained in the `com.collation.proxy.api.client.CMDBApi` class, which communicates with an RMI ApiServer on the TADDM server.

Set the `com.collation.home` property to `$COLLATION_HOME/sdk` in the SDK distribution root directory. Using the Java command line, you can set the property as follows:

```
java -Dcom.collation.home=$COLLATION_HOME/sdk main_classname
```

**Before you start using the Java API**

**Before you begin**

Before you start creating Java applications, you must verify that your development environment is properly configured.

**Procedure**

To verify your environment, complete the following steps:
1. Verify that the environment variables are properly set, and configure the `com.collation.proxy.api.port` property.
   
   See the section on configuring the SDK for more information about setting the environment variables and specific configuration parameters.

2. Verify that the TADDM server is running.
   
   See the section on verifying the SDK installation for more information.

**Exploring a sample Java application**

This section describes how to create, compile, and run a simple Java application.

**Procedure**

To create the Java sample application, complete the following steps:
1. Copy the following Java code into a file called `FindXmlExample.java`.
   
   The source code is also available in the `$COLLATION_HOME/sdk/examples/java` directory.

   ```java
   package com.collation.proxy.api.examples.java;
   // package com.collation.proxy.api.examples.java;

   import com.collation.proxy.api.client.ApiConnection;
   import com.collation.proxy.api.client.ApiException;
   import com.collation.proxy.api.client.ApiSession;
   import com.collation.proxy.api.client.CMDBApi;
   import com.collation.proxy.api.client.DataResultSet;
   import com.ibm.cdb.api.ApiFactory;

   /**
   * Simple CMDB API findXML() example:
   */
   ```
public class FindXmlExample {

    public static void main(String[] args) {

        CMDBApi api = null;
        ApiSession sess = null;
        try {
            /* Establish connection to api server
             * <p> ApiConnection.getConnection(host, port,
             *      trustoreLocation, useSSL)
             */
            ApiConnection conn = 
                    ApiFactory.getInstance().getApiConnection("localhost", -1,
                    null, false);

            /* Get a session:
             * <p> ApiSession.getSession(connection, username,
             *         password, version)
             */
            sess = ApiFactory.getInstance().getSession(conn, "smartoperator",
                    "foobar", ApiSession.DEFAULT_VERSION);

            /* Get an CMDBApi instance
             */
            api = sess.createCMDBApi();

            System.out.println("all machines which have more than 1 CPU:");
            String query = "select * from ComputerSystem where numCPUs>1";

            /* Find all of the ComputerSystem have more than 1 CPU.
             * The method: findXml(query, depth, indent, mssGuid, permissions)
             * is deprecated, as the result set may be too large to fit into
             * memory. Instead, using cursors is encouraged:
             */
            DataResultSet data = api.executeQuery(query, null, null);
            while (data.next()) {
                System.out.println(data.getXML(4));
            }
            data.close();

            System.out.println("all Oracle instances:");
            query = "select * from OracleInstance"
            data = api.executeQuery(query, null, null);
            while (data.next()) {
                System.out.println(data.getXML(4));
            }
            data.close();

        } catch (ApiException ae) {
            System.err.println("api exception:" + ae);
            ae.printStackTrace();
        } catch (Exception ex) {
            System.err.println("exception:" + ex);
            ex.printStackTrace();
        } finally {
            try {
                if (api != null) {
                    api.close();
                }
                if (sess != null) {
                    sess.close();
                }
            }
        }
    }
}
2. By default, the sample program connects to a TADDM server on the local host. If you wish to connect to a remote server, change the following line:

```java
ApiConnection conn = ApiFactory.getInstance().getApiConnection("localhost", -1, null, false);
```

For example, to connect to a server named taddmhost.ibm.com using the default ports:

```java
ApiConnection conn = ApiFactory.getInstance().
    getApiConnection("taddmhost.ibm.com",-1, null, false);
```

By default, the sample program creates a session with a user ID of smartoperator and a password of foobar. Change this to match a user ID and password that is defined to your TADDM server. For example:

```java
sess = ApiFactory.getInstance().
    getSession(conn, "administrator", "collation",ApiSession.DEFAULT_VERSION);
```

3. To compile the sample program, along with the other sample Java programs, complete the following steps:
   a. On UNIX systems:
      1) Change to the $COLLATION_HOME/sdk/examples/java directory.
      2) Make a build.sh executable:
         ```bash
         chmod +x build.sh
         ```
      3) Run the build command:
         ```bash
         ./build.sh
         ```
   b. On Windows:
      1) Change to the %COLLATION_HOME%\sdk\examples\java directory.
      2) Run the build command:
         ```bash
         build.bat
         ```

   **Note:** If the SDK is installed separately from the TADDM server, make sure that javac is already in the path and available.

4. Run the Java application using a command similar to the following example command:

```
% java -Dcom.collation.home=$COLLATION_HOME/sdk FindXmlExample
```

This command runs the sample program and retrieves the XML data from the TADDM server.

**Details about the sample Java application**

This section describes the operation of the FindXmlExample.java example.

```java
/*
 * Establish connection to API server
 */
ApiConnection conn = ApiFactory.getInstance().
    getApiConnection("localhost", -1, null, false);
```
This segment creates a new ApiConnection object to the API server, which is used as a handle to manage the API session between the client program and the TADDM server. The arguments are represented in the following list:

- The host argument is the system on which the TADDM server is running.
- The second argument is the server port. A value of -1 specifies the default port as configured by com.collation.proxy.api.port in the $COLLATION_HOME/sdk/etc/collation.properties property file.
- The third and the fourth arguments control the SSL access.

```java
/*
 * Get a session
 *<p> ApiSession.getSession(connection, username, password,
 *     version)
 */
 sess = ApiFactory.getInstance().
    getSession( conn, smartoperator, foobar,ApiSession.DEFAULT_VERSION);
/*
 * Get an CMDBApi instance
 */
 api = sess.createCMDBApi();
```

This segment connects the CMDBApi object to the TADDM server. If you need to connect to multiple servers, in a large scale distributed data center scenario for example, you can use multiple CMDBApi objects with each maintaining context to a specific TADDM server instance.

```java
System.out.println("all machines which have more than 1 CPU:");
String query = "select * from ComputerSystem where numCPUs>1";
/*
 * Find all of the ComputerSystem have more than 1 CPU.
 * The method: findXml(query, depth, indent, mssGuid, permissions)
 * is deprecated, as the result set may be too large to fit into
 * memory. Instead, using cursors is encouraged:
 */
 DataResultSet data = api.executeQuery(query, null, null);
```

This segment uses an initialized CMDBApi object to retrieve data from the TADDM server using the executeQuery method. The following describes the arguments:

- The first argument specifies the query. See the MQL query elements in the section on Introducing the Model Query Language for more information.
- The second argument is the Management Software Systems (MSS) GUID. A value of null indicates that the results are returned from the query for all records regardless of the MSS they are associated with.
- The third argument is the permissions array. Permissions are supplied during data-level access control configuration. The permissions supplied here should match those used to configure data-level access control. For example, supplying an "Update" permission would constrain the returned objects to those the caller has the authority to update per the data-level access control configuration. A value of null indicates that all objects the user has access to will be returned.

**Related reference:**

"Management Software Systems" on page 35

MSS methods manage Management Software Systems in the Common Data Model. You can use the Management Software System methods to register and delete an MSS in the Common Data Model. You can also retrieve information about the Management Software Systems that have been registered with the TADDM database.
Best practices

This section describes the following best practices when using the Java API:

- Optimizing data access
- Following links between model objects

Optimizing data access

As with other data access APIs, the TADDM API can return large amounts of data, potentially overwhelming system resources. Therefore, avoid retrieving all data in large environments. This method requires frequent synchronization with the TADDM topology to ensure that changes in the topology are captured.

The following options give you some different methods to retrieve your data:

- A suggested pattern of usage is to only retrieve the elements and identities, and not necessarily the detailed configuration data. This limits the amount of data that is transferred. When you need the detailed configuration of an element, you can make a subsequent findChanges() call using the object ID as a parameter.
- Perform incremental change data access. This method requires that you use the following type of call to the findChanges() method:

  ```java
  findChanges(java.lang.String root, java.lang.String query, int depth,
              long start, long end, int changeType)
  ```

  The `start` and `end` parameters specify a time range, while the `changeType` parameter specifies Created, Deleted or Updated. This findChanges() call returns only those objects which are of the type specified using the `changeType` parameter within the given time range. Use this method when performing incremental synchronization of topology data after a full baseline data transfer.
- The find method returns all data at once, which can cause memory usage issues. To avoid this problem, use the executeQuery method to scroll through data using cursors:

  ```java
  DataResultSet rs = api_.executeQuery("select * from ComputerSystem", null, null);
  while (rs.next()){
    ...
  }
  rs.close();
  ```

- Use the findCount method to efficiently count the number of objects matching a query:

  ```java
  long count = api.findCount("select * from ComputerSystem", null);
  ```

Larger memory settings for the Topology Manager

If you are running non-specific queries against large databases, you can encounter memory issues on the TADDM server and the API client. Use specific queries to identify the size of the result sets and memory requirement.

For more generic queries that generate a large result set, more memory must be allocated. Use this method if you receive an out of memory error message.

To increase the available memory, use the following values:

- On the TADDM server, you can increase the available memory for the topology manager virtual machine to 2048 MB by editing one of the following configuration files:
  - Domain server: cmdb-context.xml
- Synchronization server: ecmdb-context.xml

  In the appropriate file, locate the jvmArgs property for the TopologyManager virtual machine and change the -Xmx value to 2048M.

- On the API client, increase the memory specified in the api.sh or api.bat file to 2048M.

**Following links between model objects**

Most data elements in the Common Data Model are stand-alone. In many cases, links between model objects, such as LogicalDependency, are represented by storing object IDs, which the TADDM API does not automatically follow. In these cases, you must apply additional logic within your client application.

**Java API Method summary**

Using the Java API, you can create applications that add, update, and delete model objects. You can query model objects by class name or object ID number. You can also use the interface to manage relationships between objects, perform comparisons, and examine the change history in the TADDM database.

The Java API can be summarized using the following categories that are presented in more detail in their appropriate sections:

- Managing sessions
- Discovery management
- Managing the model
- Find, update, and delete operations
- Managing collections
- Managing relationships
- Metadata
- Management Software Systems
- MSSObjectLink
- Change history
- Presentation
- Managing versions
- Security
- Creating and managing access lists
- Managing application templates

**Change history:**

Change history methods determine the change history within the Common Data Model. You can use the change history methods to retrieve the change history for managed elements within the Common Data Model. You can also trigger the propagation and calculation of the change history, as required.

Table 9 describes the change history methods you can use.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getChangeHistory(Guids, start, end)</td>
<td>Return the change history for the specified array of GUIDs.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| `getChangeHistory(Guids, start, end, filterType)` | Get the change history for multiple GUIDs, filtered by the filterType parameter. This method is like the `getChangeHistory()` with the addition of filterType, which can be set to the following values:  
  - DataApi.CREATED  
  - DataApi.DELETED  
  - DataApi.UPDATED  
  - DataApi.UPDATECREATE  
  - DataApi.ANYCHANGE |
| `getChangeHistory(Guid, start, end)`             | Return the change history for the specified GUID.                                                                                                                                                           |
| `getChangeHistory(classNames, start, end)`       | Return the change history for the specified classes.                                                                                                                                                       |
| `getChangeHistoryByPersistTime(classNames, start, end)` | Return the change history for the specified classes based on persist time.                                                                                                                                   |
| `getChangeHistoryInXML(Guids, start, end)`       | Return the change history for the specified array of object IDs.                                                                                                                                            |
| `getChangeHistoryInXML(Guids, start, end, filterType)` | Get the change history for multiple GUIDs, filtered by filterType. This method is like the `getChangeHistory()` with the addition of filterType, which can be set to the following values:  
  - DataApi.CREATED  
  - DataApi.DELETED  
  - DataApi.UPDATED  
  - DataApi.UPDATECREATE  
  - DataApi.ANYCHANGE |
| `getChangeHistoryInXML(Guid, start, end)`        | Return the change history for the specified GUID.                                                                                                                                                           |
| `getPropagatedChanges(primaryKey)`               | Get the root causes for a given change history, returned as an XML representation of ChangeHistory objects that caused the actual change. For instance, if an Apache Server module changes, the changes are propagated to the top-level Apache Server. You can use this method to determine the cause that triggered the Apache Server to be changed. |
| `getPropagatedChangesInXML(primaryKey)`          | Get the root causes for a given change history, returned as an XML representation of ChangeHistory objects that caused the actual change. For instance, if an Apache Server module changes, the changes are propagated to the top-level Apache Server. You can use this method to determine the cause that triggered the Apache Server to be changed. |
Table 9. Change history methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processChanges()</td>
<td>Trigger propagation and calculation of the change history since the last discovery or the last time that processChanges() was called. Without it, changes are not calculated until the next time a discovery is run. This method must be used for isolated changes. For multiple updates, such as bulkload operations, use the startBulkload() and endBulkload() methods.</td>
</tr>
<tr>
<td>getChangeHistory(classNames, start, end)</td>
<td>Return the change history for the specified classes.</td>
</tr>
<tr>
<td>getChangeHistoryByPersistTime(classNames, start, end)</td>
<td>Return the change history for the specified classes, based on persist time.</td>
</tr>
</tbody>
</table>
| getChangedClasses(start, end, changeType) | Return an array of class types that have changed between the start and end dates. The specified change type can be any of the following:  
  - DataApi.CREATED  
  - DataApi.DELETED  
  - DataApi.UPDATED  
  - DataApi.UPDATECREATE  
  - DataApi.ANYCHANGE |
| getChangedClassesForDeltaSynching(start, end, ChangeType) | Returns an array of class types that have changed between the start and end dates, based on persist time. The specified change type can be any of the following:  
  - DataApi.CREATED  
  - DataApi.DELETED  
  - DataApi.UPDATED  
  - DataApi.UPDATECREATE  
  - DataApi.ANYCHANGE |
| getChangeHistory(guids, start, end, offset, nextBatch) | Returns the change history for the specified list of GUIDs. The nextBatch parameter specifies the size of the batch of records to be returned, starting at the specified offset. This allows for a scalable approach to returning change history information. |

Discovery management:

Discovery methods manage discovery runs. You can use the discovery methods to start and stop discoveries, and enable and disable update events. You can also use the methods to get the status of a discovery, and clear all discovery elements from the topology.

Table 10 describes the discovery methods you can use.

Table 10. Discovery methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortDiscovery()</td>
<td>End the current discovery run.</td>
</tr>
</tbody>
</table>
Table 10. Discovery methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| getStatus() | Return the status of the current discovery run. The status can be any of the following values:  
- Running  
- Idle  
- Waiting  
- Aborted |
| startDiscovery(scope, runName) | Start a new discovery with a given scope. The scope can be a name, or contain IP ranges, networks, and addresses, with includes and excludes of specific IP addresses. |
| startDiscovery(runDef, runName) | Start a new discovery based on a discovery run definition that specifies profile, scope, and run name information. |
| startDiscover(guidList, runName) | Start a rediscovery of the objects with the specified globally unique identifiers (GUIDs). |
| abortDiscovery(runId) | End the specified discovery run. |

Find operations:

Find methods access objects in the Common Data Model. You can use the find methods to return model objects matching a specific query or return information about specific managed elements. You can also use the methods to return objects that have changed during a specified time period.

Table 11 describes the find operations you can perform.

Note: Many of the find methods using a depth parameter are now deprecated, because they do not scale well when querying large amounts of data. If you need to query data at a depth greater than one, use an executeQuery method. Each executeQuery method returns a DataResultSet object from which you can retrieve information about model objects, and you can use a cursor to scroll through the data.

Table 11. Find methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find(guid, depth, mss, permissions)</td>
<td>Return information about a specific managed element.</td>
</tr>
<tr>
<td>find(guid, depth, permissions)</td>
<td>Same as find(String, int, Guid), except a specific object instance is searched by GUID rather than an entire set of objects using a query.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>find(query, depth, mss, permissions)</td>
<td>Return model objects matching the specified query. Note the following details about the depth parameter:</td>
</tr>
<tr>
<td>Note: This method is deprecated. Use executeQuery or a find method with a fillFlag parameter.</td>
<td>・ A depth of 0 returns an object with only its GUID set.</td>
</tr>
<tr>
<td></td>
<td>・ A depth of 1 returns all top-level attributes, along with contained objects having only their GUID attributes set.</td>
</tr>
<tr>
<td></td>
<td>・ Setting depth to DEPTH_INFINITE recursively locates all attributes until no additional objects are found. Cycles are avoided.</td>
</tr>
<tr>
<td>public ModelObject[] find(String query,</td>
<td>Return model objects matching the specified query.</td>
</tr>
<tr>
<td>boolean fillFlag, Guid mss, String[]</td>
<td>• query - Model language query to select and filter results.</td>
</tr>
<tr>
<td>permissions)</td>
<td>• fillFlag – Whether to populate all of the attributes.</td>
</tr>
<tr>
<td></td>
<td>• mss - Managed Software System ID, or null if none</td>
</tr>
<tr>
<td></td>
<td>• permissions - Optional list of permissions to restrict results</td>
</tr>
<tr>
<td>findChanges(query, depth, start, end,</td>
<td>Return objects that have changed during the specified period for a given change type.</td>
</tr>
<tr>
<td>changeType)</td>
<td>Note: This method is deprecated. Use executeQuery or a find method with a fillFlag parameter.</td>
</tr>
<tr>
<td></td>
<td>public ModelObject[] findChanges(String query, boolean fillFlag, long start, long end, int changeType)</td>
</tr>
<tr>
<td></td>
<td>Return objects that have changed during the specified period for a given change type.</td>
</tr>
<tr>
<td>findChangesInXml(query, depth, start, end,</td>
<td>Return objects that have changed in the specified period for a given change type.</td>
</tr>
<tr>
<td>changeType)</td>
<td>public String findChangesInXml(String query, boolean fillFlag, long start, long end, int changeType)</td>
</tr>
<tr>
<td></td>
<td>findCollections(guid, permissions) See the section on Managing collections.</td>
</tr>
<tr>
<td>findImpactedBusinessApplications(objects)</td>
<td>See the section on Presentation.</td>
</tr>
<tr>
<td>findImpactedBusinessServices(objects)</td>
<td>See the section on Presentation.</td>
</tr>
<tr>
<td>findJDO(root, jdoQuery, jdoVarDecl, depth,</td>
<td>Same as find(String, int, Guid) except that the query must contain a Java Data Object query (JDOQL), with optional variable declarations.</td>
</tr>
<tr>
<td>mss, permissions)</td>
<td>Note: This method is deprecated. Use findJDO(root, jdoQuery, jdoVarDecl, fillFlag, mss, permissions).</td>
</tr>
<tr>
<td></td>
<td>public ModelObject[] findJDO(String root, String jdoQuery, String jdoVarDecl, boolean fillFlag, Guid mss, String[] permissions)</td>
</tr>
<tr>
<td></td>
<td>Same as find(String, int, Guid) except that the query must contain a Java Data Object query (JDOQL), with optional variable declarations.</td>
</tr>
<tr>
<td></td>
<td>findRelationships(managedElementGuid, direction, type, scope, permissions) See the section on Presentation.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Relationship[] findRelationships(Guid[] sourceGuids, Guid[] targetGuids, String[] types, int comparisonFlags) throws ApiException;</td>
<td>Retrieves relationships with only basic information returned.</td>
</tr>
<tr>
<td>findXML(query, depth, indent, mss, permissions)</td>
<td>Return an XML document containing a list of all objects matching the specified query. This method is similar in function to find(String, int, Guid) except that the objects are converted to XML format.</td>
</tr>
<tr>
<td>findXML(Guid, indent, depth, permissions)</td>
<td>Same as findXML(String, int, int, Guid) except that a specific object instance is searched by ID, rather than a whole set of objects using a query.</td>
</tr>
<tr>
<td>public String findXML(String query, boolean fillFlag, int indent, Guid mss, String[] permissions)</td>
<td>Same as findXML(String, int, int, Guid) except that a specific object instance is searched by ID, rather than a whole set of objects using a query.</td>
</tr>
<tr>
<td>public long findCount(String query, String[] permissions)</td>
<td>Returns a count of the objects matched by the specified MQL query string.</td>
</tr>
<tr>
<td>executeQuery(query, mss, permissions)</td>
<td>Executes an MQL query and returns a scrolling cursor interface.</td>
</tr>
<tr>
<td>executeQuery(query, defaultDepth, mss, permissions)</td>
<td>Executes an MQL query and returns a scrolling cursor interface.</td>
</tr>
<tr>
<td>findChangesForDeltaSynching(query, fillFlag, start, end, changeType)</td>
<td>Returns objects that were persisted in the data store during the specified period of time for the specified change type.</td>
</tr>
</tbody>
</table>

**Management Software Systems:**

MSS methods manage Management Software Systems in the Common Data Model. You can use the Management Software System methods to register and delete an MSS in the Common Data Model. You can also retrieve information about the Management Software Systems that have been registered with the TADDM database.

Table 12 describes the MSS methods you can use.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteManagementSoftwareSystem(guid)</td>
<td>Delete a Management Software System (MSS) from the TADDM database. Objects and relationships owned or discovered solely by this MSS are also deleted. However, objects and relationships owned or discovered by this and another other MSS are not deleted. Only the association between this MSS and the objects and relationships owned by it are deleted.</td>
</tr>
</tbody>
</table>
Table 12. Management Software System methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getManagementSoftwareSystemLinks(guid, mss, permissions)</td>
<td>Return the source tokens for a managed element or relationship. If a Management Software System is not specified, the method returns the source tokens for each Management Software System that either discovered or owns the specified object.</td>
</tr>
<tr>
<td>getManagementSoftwareSystems(guid, permissions)</td>
<td>Get an array of Management Software Systems that have been registered with the TADDM database. Optionally, the GUID of a managed element or relationship can be provided to return only Management Software Systems that own the managed element or relationship.</td>
</tr>
</tbody>
</table>
| registerManagementSoftwareSystem(mss) | Register a new Management Software System with the TADDM database. Note the following details:  
  - This method only inserts an object. The method does not replace or update an existing object.  
  - The model must contain keys for the object.  
  - When a key refers to a parent object, the parent object must exist.  
  - When the object directly references a GUID, it is the developer's responsibility to ensure that the GUID exists in the TADDM database. |
| updateManagementSoftwareSystem(mss) | Update or insert a Management Software System into the database. Note the following details:  
  - This method can be used to insert, replace, or update an existing object.  
  - The model must contain keys for the object.  
  - When a key refers to a parent object, the parent object must exist.  
  - When the object directly references a GUID, it is the developer's responsibility to ensure that the GUID exists in the TADDM database. |

MSSObjectLink:

MSSObjectLink is an association between a Management Software System and a managed element that it owns.

The MSSObjectLink methods can be used to retrieve all the MSSObjectLinks for the given MSS and managed elements. These MSSObjectLinks represent either all the managed elements owned by a given MSS, or all the MSSs that own a given managed element. From this release forward, an MSSObjectLink will no longer be stored as a model object, therefore you will no longer be able to use the find API
to get the MSSObjectLink objects. Instead, use the new APIs described in Table 13 along with the getManagementSoftwareSystemLinks API.

**Table 13. MSSObjectLink**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getObjectSourceSystems(Guid[] objectId, String[] permission)</td>
<td>Returns all the MSS that owns the managed elements or relationships. This method returns an array of MSSObjectLink that corresponds to the MSS that owns a managed element. You can pass in a maximum of 50 managed elements at a time in the Guid array.</td>
</tr>
<tr>
<td>getObjectSourceSystems(Guid objectId, Guid mssGuid, String[] permission)</td>
<td>Returns an array of MSSObjectLinks where each MSSObjectLink corresponds to the source token of an object owned by the given MSS. If only the mssGuid is specified, this method will return the source tokens of all the objects that it owns. On the other hand, if you specify only the object guid, it will return the source tokens of all the MSSs that own it.</td>
</tr>
<tr>
<td>getObjectSourceSystems(Guid mssGuid, String mssSourceToken)</td>
<td>Returns an MSSObjectLink that corresponds to a managed element that is owned by a given MSS and identified by its source token (source token must be specified).</td>
</tr>
<tr>
<td>getObjectSourceSystems(String joinQuery)</td>
<td>Returns an array of MSSObjectLinks that satisfy the JOIN query between MSSObjectLink and any other model objects. From this release onwards, since MSSObjectLinks will be stored directly as relational data rather than as a model object, you won't be able to use MQL to JOIN MSSObjectLink with any other model objects. The joinQuery that is passed in this method will be an SQL query instead. The attributes that are specified in the SELECT clause of the SQL query must be the attributes of the MSSObjectLink object since this method returns an array of MSSObjectLink objects. Example: Get all the ComputerSystems that are owned by a given MSS.</td>
</tr>
</tbody>
</table>

```sql
SELECT mssobjlink_rel.obj_x,
       mssobjlink_rel.mssourcetoken_x,
       mssobjlink_rel.guid_x,
FROM mssobjlink_rel, compsys
WHERE mssobjlink_rel.obj_x = compsys.guid_x
AND mssobjlink_rel.mss_x='5D65T789UK3'
```
Managing access lists:

The access list methods create and manage access list entries from the Java API. Third-party applications can manage identities using these methods.

The following tasks can be completed using these API methods:

**Create and delete the access list entry**
You can create and delete the access list entry to maintain them automatically.

**Update the properties of the access list entry**
You can update the properties of the access list entry. The API can be used to change the password.

**Get the properties of the access list entry**
You can get the particular properties of the access list entry shown in Access List window of the Discovery Management Console. The API cannot be used to retrieve the password.

**Verify the property value of the access list entry**
You can verify whether the given property value matches the property value in the existing access list entry. This API can be used to verify the password of the access list entry.

Table 14 describes the access list methods you can use.

**Table 14. Access list methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void deleteDiscoveryAccessEntry(java.lang.String authClassName, java.lang.String name)</code></td>
<td>Delete the access entry with the specified class and name.</td>
</tr>
<tr>
<td><code>DiscoveryAccessEntry[] getAllDiscoveryAccessEntries()</code></td>
<td>Get all the discovery access entries.</td>
</tr>
<tr>
<td><code>DiscoveryAccessEntry getDiscoveryAccessEntry(java.lang.String authClassName, java.lang.String name)</code></td>
<td>Get the discovery access entry with the specified class and name.</td>
</tr>
<tr>
<td><code>DiscoveryAccessEntry updateDiscoveryAccessEntry(DiscoveryAccessEntry discoveryAccess)</code></td>
<td>Update the properties of the access entry with the specified class and name.</td>
</tr>
<tr>
<td><code>boolean verifyDiscoveryAccessEntry(DiscoveryAccessEntry discoveryAccess)</code></td>
<td>Verify whether the given properties match the current properties of the access entry with the specified class and name.</td>
</tr>
</tbody>
</table>

**Note:** Access list entries registered in the discovery profile are not supported by the API provided.

Example Java code

1. The following example Java code shows how to create the connection, session, and API instance:

   ```java
   CMDBApi api;
   try {
     ApiConnection conn_ = ApiFactory.getInstance().getApiConnection("localhost", -1,null,false);
     ApiSession session_ = ApiFactory.getInstance().getSession(conn_, user, password, CMDB_DEFAULT);
     api = session_.createCMDBApi();
   } catch (ApiException ex) {
   ```
2. The following example Java code shows how to programmatically create discovery access entries:

```java
try {
    // IBM WebSphere access entry
    DiscoveryAccessEntry entry = new DiscoveryAccessEntry
        (DiscoveryAccessEntry.AUTHCLASS_WEBSPHERE, "user3-websphere");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope3");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 3);
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "wasadmin");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_TRUSTSTOREFILECONTENTS, new byte[]{0x10, 0x20, 0x30});
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_TRUSTSTOREPASSPHRASE, "password");
    entry = api.updateDiscoveryAccessEntry(entry);

    // SNMP access entry
    entry = new DiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_SNMP, "user4-snmp");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope4");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 4);
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_COMMUNITYSTRING, "public");
    entry = api.updateDiscoveryAccessEntry(entry);

    // SNMPv3 access entry
    entry = new DiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_SNMPV3, "user5-snmpv3");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope5");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 5);
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "snmp");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_AUTHPROTOCOL, "MD5");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PRIVPASSWORD, "privpassword");
    entry = api.updateDiscoveryAccessEntry(entry);

    // Cisco access entry
    entry = new DiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_CISCO, "user6-cisco");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope6");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 6);
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "cisco");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
    entry.setProperty("enablepassword", "enablepassword");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ENABLEPASSWORD, "enablepassword");
    entry = api.updateDiscoveryAccessEntry(entry);

    // ccmserver access entry
    entry = new DiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_CCMSSERVER, "user7-sapccms");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope7");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 7);
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "ccms");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_CLIENTID, "clientid");
    entry = api.updateDiscoveryAccessEntry(entry);

    // Computer system access entry
    entry = new DiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_HOST, "user1-host");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope1");
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 1);
    entry = api.updateDiscoveryAccessEntry(entry);
}
```

entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 1);
entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "root");
entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
entry = api.updateDiscoveryAccessEntry(entry);

// Windows computer system access entry
entry = new DiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_WINDOWSHOST, "user2-winhost");
entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope2");
entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "Administrator");
entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
entry.setProperty(DiscoveryAccessEntry.PROPERTY_AUTHTYPE, "authType_default");
entry = api.updateDiscoveryAccessEntry(entry);
}

try {
    DiscoveryAccessEntry entry;
    DiscoveryAccessEntry[] entries = api.getAllDiscoveryAccessEntries();
    for (int i = 0; i < entries.length; i++) {
        entry = entries[i];
        String authClassName = (String) entry.getAuthClassName();
        Integer order = (Integer) entry.getProperty(DiscoveryAccessEntry.PROPERTY_ORDER);
        switch (order.intValue()) {
            case 1:
                // DiscoveryAccessEntry.AUTHCLASS_HOST.equals(authClassName));
                break;
            case 2:
                // DiscoveryAccessEntry.AUTHCLASS_WINDOWSHOST.equals(authClassName));
                break;
            case 3:
                // DiscoveryAccessEntry.AUTHCLASS_WEBSPHERE.equals(authClassName));
                break;
            case 4:
                // DiscoveryAccessEntry.AUTHCLASS_SNMP.equals(authClassName));
                break;
            case 5:
                // DiscoveryAccessEntry.AUTHCLASS_SNMPV3.equals(authClassName));
                break;
            case 6:
                // DiscoveryAccessEntry.AUTHCLASS_CISCO.equals(authClassName));
                break;
            case 7:
                // DiscoveryAccessEntry.AUTHCLASS_CCMSERVER.equals(authClassName));
                break;
            default:
                break;
        }
    }
} catch (ApiException ae) {
    System.err.println("api exception:" + ae);
} catch (Exception ex) {
    System.err.println("exception:" + ex);
}

3. The following example Java code shows how to get all discovery access entries:

try {
    DiscoveryAccessEntry entry;
    DiscoveryAccessEntry[] entries = api.getAllDiscoveryAccessEntries();
    for (int i = 0; i < entries.length; i++) {
        entry = entries[i];
        String authClassName = (String) entry.getAuthClassName();
        Integer order = (Integer) entry.getProperty(DiscoveryAccessEntry.PROPERTY_ORDER);
        switch (order.intValue()) {
            case 1:
                // DiscoveryAccessEntry.AUTHCLASS_HOST.equals(authClassName));
                break;
            case 2:
                // DiscoveryAccessEntry.AUTHCLASS_WINDOWSHOST.equals(authClassName));
                break;
            case 3:
                // DiscoveryAccessEntry.AUTHCLASS_WEBSPHERE.equals(authClassName));
                break;
            case 4:
                // DiscoveryAccessEntry.AUTHCLASS_SNMP.equals(authClassName));
                break;
            case 5:
                // DiscoveryAccessEntry.AUTHCLASS_SNMPV3.equals(authClassName));
                break;
            case 6:
                // DiscoveryAccessEntry.AUTHCLASS_CISCO.equals(authClassName));
                break;
            case 7:
                // DiscoveryAccessEntry.AUTHCLASS_CCMSERVER.equals(authClassName));
                break;
            default:
                break;
        }
    }
} catch (ApiException ae) {
    System.err.println("api exception:" + ae);
} catch (Exception ex) {
    System.err.println("exception:" + ex);
}

4. The following example Java code shows how to get a specific discovery access entry:

try {
    DiscoveryAccessEntry entry = api.getDiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_SNMP, "user4-snmp");
    String authClassName = (String) entry.getAuthClassName();

5. The following example Java code shows how to update a specific discovery access entry:

```java
try {
    // Create an entry with the same name as an existing entry
    DiscoveryAccessEntry entry = new DiscoveryAccessEntry
        (DiscoveryAccessEntry.AUTHCLASS_HOST, "user1-host");

    // Change the scope
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_SCOPENAME, "scope1c");

    // Change the order
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 2);

    // Change the username
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "rootc");

    // Change the password
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "passwordc");

    // Update the entry
    entry = api.updateDiscoveryAccessEntry(entry);
} catch (ApiException ae) {
    System.err.println("api exception:" + ae);
} catch (Exception ex) {
    System.err.println("exception:" + ex);
}
```

6. The following example Java code shows how to verify a discovery access entry:

```java
try {
    // Create an entry with the same name as the existing entry to verify
    DiscoveryAccessEntry entry = new DiscoveryAccessEntry
        (DiscoveryAccessEntry.AUTHCLASS_HOST, "user1-host");

    // Set the order property to the wrong number
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 1);
    boolean result = api.verifyDiscoveryAccessEntry(entry));

    // Set the order property to the correct number
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_ORDER, 2);
    result = api.verifyDiscoveryAccessEntry(entry));

    // Set the password property to the wrong value
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
    result = api.verifyDiscoveryAccessEntry(entry));

    // Set the password property to the correct value
    entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "passwordc");
    result = api.verifyDiscoveryAccessEntry(entry));

    // This result should be true
    result = api.verifyDiscoveryAccessEntry(entry));
} catch (ApiException ae) {
```
```java
try {
    api.deleteDiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_HOST, "user1-host");
} catch (ApiException ae) {
    System.err.println("api exception:" + ae);
} catch (Exception ex) {
    System.err.println("exception:" + ex);
}
```

7. The following example Java code shows how to delete a discovery access entry:

```java
try {
    api.deleteDiscoveryAccessEntry(DiscoveryAccessEntry.AUTHCLASS_HOST, "user1-host");
} catch (ApiException ae) {
    System.err.println("api exception:" + ae);
} catch (Exception ex) {
    System.err.println("exception:" + ex);
}
```

Managing collections:

Collection methods manage collections in the Common Data Model. You can use the collection methods to add and remove members in a collection, and to retrieve all collections to which a specific managed element belongs.

Table 15 describes the collection methods you can use.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCollectionMembers(collectionGuid, guids)</td>
<td>Add members to a collection. This method is unavailable in the Enterprise JavaBeans (EJB) interface.</td>
</tr>
<tr>
<td>findCollections(guid, permissions)</td>
<td>Retrieve all collections to which the specified managed element belongs. A collection can contain other collections, but this method returns only the first-level of collection to which the specified managed element belongs.</td>
</tr>
<tr>
<td>removeCollectionMembers(collectionGuid, guids)</td>
<td>Remove members from a collection. Members of a collection can be removed by specifying an array of collection GUID attributes or an array of managed element GUID attributes. When a member to be removed is not currently a member of the collection, the member is ignored.</td>
</tr>
</tbody>
</table>

Managing the model:

Model methods manage objects in the Common Data Model. You can use the model methods to add, delete, and update objects within the Common Data Model. You can also use the methods to compare objects, and to rebuild the derived data, such as dependencies, relationships and data consolidation.

Table 16 describes the model methods you can use.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(obj, mss)</td>
<td>Add a new object to the database.</td>
</tr>
<tr>
<td>addArrayElements(object, attrName, elements, mss)</td>
<td>Add the elements to the named array of the specified object without fetching either the object or the array.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>compare(left, right, opts)</td>
<td>Compare a model object (or golden master) against a set of objects.</td>
</tr>
<tr>
<td>compare(obj1, objs, opts)</td>
<td></td>
</tr>
</tbody>
</table>
| delete(guids, mss)   | Delete the objects specified by the GUID from the database and cascade delete all objects contained within the objects in either of the following cases:  
<p>| delete(obj, mss)     | • No Management Software System (MSS) is provided.                                               |
|                      | • The object is owned exclusively by the specified MSS.                                          |
|                      | The object is not deleted when an MMS is provided and the object is owned by another MSS. Instead, the association between the object and MSS is deleted. |
|                      | When an object is deleted from the TADDM database, all relationships and collection memberships associated with the object are also deleted. |
|                      | If the specified object is a top-level model object, all contained objects are also removed. For example, when a computer system is removed, the operating system and IP interfaces contained within the object, along with the relationships between the computer system and IP interfaces, are also removed. |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteStale(mss, date)</td>
<td>Delete managed elements and relationships that have not been touched since a specified date. The managed elements and relationships that have an update time stamp less than or equal to the specified date are deleted. If a stale managed element or relationship is owned by more than one management software system, the managed element or relationship is not deleted from the database. Only the association is deleted between the managed element or relationship and the specified management software system. However, if a stale managed element or relationship is owned only by the specified management software system, the managed element or relationship is deleted from the database. All relationships and collection memberships associated with a deleted managed element are also deleted. If the specified object is a top-level model object, all contained objects are also removed. For example, when a computer system is removed, the operating system and IP interfaces contained in the object, along with the relationships between the computer system and IP interfaces, are also removed.</td>
</tr>
<tr>
<td>endBulkload(transactionId)</td>
<td>Mark the end of a bulkload operation. Each caller that calls the startBulkload() method must call endBulkload() to release the lock on the storage subsystem.</td>
</tr>
<tr>
<td>exportData(directoryToWriteTo, maxSize, mss)</td>
<td>Export all objects in the TADDM database to the specified directory, creating files for each object class using the find() method with infinite depth. When maxSize bytes are exceeded, a new file is created using a .N extension, with N incremented as required. The format of the output adheres to the XML schema format.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>importData(source, rebuildTopo, mss)</code></td>
<td>Convert XML data from the specified source into model objects which are updated within the TADDM database, according to the following rules:</td>
</tr>
<tr>
<td></td>
<td>• When the source is a file, the contents of the single file are read and inserted.</td>
</tr>
<tr>
<td></td>
<td>• When the source is a directory, each file in the directory is imported.</td>
</tr>
<tr>
<td></td>
<td>• When the source is a remote object, such as an HTTP address, each update operates within its own transaction.</td>
</tr>
<tr>
<td></td>
<td>Errors roll back the update of the current object only, and the import then proceeds.</td>
</tr>
<tr>
<td><code>rebuildTopology()</code></td>
<td>Rebuild the TADDM database derived data, such as dependencies, relationships and data consolidation. During this operation, the entire database is locked against updates.</td>
</tr>
<tr>
<td><code>removeArrayElements(object, attrName, elements, mss)</code></td>
<td>Remove the specified elements of the given object from the named array in the TADDM database without fetching either the object or the array to the client.</td>
</tr>
<tr>
<td><code>startBulkload(timeOutInSeconds)</code></td>
<td>Lock the storage subsystem from other changes to the database, including discoveries and synchronizations. You must call this method before performing major updates. The lock remains until the <code>endBulkload()</code> method is called.</td>
</tr>
</tbody>
</table>
Table 16. Model management methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update(ModelObject obj, mss)</td>
<td>Update or insert a model object into the database. Attributes which are set in the source object are merged in the destination, while attributes which are not set are not updated. When an object with the specified type and key does not exist, a new object is created within the database.</td>
</tr>
</tbody>
</table>
| update(ModelObject[] obj, mss) | Note the following details:  
• The new object must have either its GUID or a key set. When a key refers to a parent object, the parent object must exist. An empty object with only the GUID set is enough to identify a parent.  
• When the source object directly references a GUID, it is the developer's responsibility that the GUID exists in the TADDM database.  
• When the mss GUID is null, the objects are inserted or updated in the TADDM database and no MSS-Object link is updated. When the mss GUID is not null, the MSS-Object link is updated.  
• It might be necessary to rebuild the topology to automatically infer dependencies and explicit relationships for the new object.  
• Arrays are replaced in their entirety. |
| updateXML(xml, mss) | Same as update(ModelObject[] obj, mss) except that the objects are represented as an XML string. |

Example

This example illustrates how to compare objects.

```java
//Find two comparable objects to compare first.
ModelObject mo[] = api.find(
   "SELECT * FROM SunSPARCUnitaryComputerSystem", 3, null, null);

if (mo != null) {
   if (mo.length > 1) {
      ModelObject mo1 = mo[0];
      ModelObject mo2 = mo[1];

      try {
         System.out.println("Comparing " + mo1.getDisplayName() + 
         " to " + mo2.getDisplayName());
      } catch (Exception e) {
         e.printStackTrace();
      }

      // ObjectRef is a simple data structure that contains the GUID and the
      // version of the object to be compared.
      ObjectRef objectRef2 = new ObjectRef(mo2.getGuid(), 0);
      ComparisonResult result = api.compare(new ObjectRef(mo1.getGuid(), 0),
```
new ObjectRef[]{objectRef2},
new CompareOptions(true));
handleModel((TreeTableModel)result);
}
}

public void handleModel(TreeTableModel model) {
    CompareResultRow row = model.getRoot();
    handleRow(model, row, "");
}

private void handleRow(
    TreeTableModel model, CompareResultRow row, String attributeName){
    System.out.println("Handling row " + row);
    int nColumns = model.getColumnCount();
    for (int i = 0; i < nColumns; i++) {
        String columnName = model.getColumnName(i);
        Object value = model.getValueAt(row, i);
        System.out.println("Col Name " + columnName + " value " + value);
        //First column, this is the attributeName
        if (i == 0) {
            if (!"".equals(attributeName)) {
                attributeName = attributeName + ":" + String.valueOf(value);
            } else {
                attributeName = String.valueOf(value);
            }
        }
        // Calculate the column name and persist to db here.
    }
    List children = row.getChildren();
    if (children != null) {
        Iterator it = children.iterator();
        while (it.hasNext()) {
            CompareResultRow resultRow = (CompareResultRow) it.next(); //recurse
            handleRow(model, resultRow, attributeName);
        }
    }
}

Managing relationships:

Relationship methods manage relationships between objects in the Common Data Model. You can use the relationship methods to add and delete relationships between managed elements in the Common Data Model. You can use this method to retrieve a relationship graph for a given relationship type.

Table 17 on page 48 describes the relationship methods you can use.
Table 17. Relationship methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRelationships(relationships, mss)</td>
<td>Add one or more relationships to the TADDM database. The source and target managed elements of a relationship must exist before adding a relationship between the elements. A relationship instance cannot exist in the TADDM database by itself. It must be discovered or owned by one or more Management Software Systems (MSS).</td>
</tr>
</tbody>
</table>
| deleteRelationships(guids, mss)             | Remove one or more relationships from TADDM database, in either of the following cases:  
  - When no Management Software System is specified  
  - When a relationship is owned only by the specified MSS  

  The relationship is not deleted when an MSS is specified and a relationship is owned by another MSS. In this case, only the association between the relationship and the MSS is deleted. |
| deleteRelationships(type, source, target, mss) |                                                                                                                                                                                                                                                                                                                                 |
| findRelationships(managedElementGuid, direction, type, scope, permissions) | Retrieve the relationship graph for the given relationship type starting from the specified managed element. The relationships that are stored in the TADDM database can be traversed in the following directions:  
  - Starting from a source managed element and going forward  
  - Starting from a target managed element and going backward  

  Relationship[] findRelationships(Guid[], sourceGuids, Guid[] targetGuids, String[] types, int comparisonFlags) throws ApiException;  

  Retrieves relationships with only basic information returned. This method runs more quickly than the findrelationships(managedElementGuid, direction, type, scope, permissions) method. |

Managing sessions:

Session methods establish connections and sessions with the server. You can use the session methods to open and close sessions with the TADDM server, and retrieve the current connection.

Table 18 describes the session methods you can use.

Table 18. Session methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close()</td>
<td>Close a session.</td>
</tr>
</tbody>
</table>
### Table 18. Session methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| ApiFactory.getInstance().getConnection(host, port, trustStoreLocation, useSSL)   | Create a connection using the specified host and port. If the specified port value is -1, the default port specified in the collation.properties file is used.  
  The trustStoreLocation parameter specifies the location of the certificate file to use for SSL connections.  |
| ApiFactory.getInstance().getSession(conn, user, password, version)     | Return a Session object, which is used to execute TADDM API methods.                                                                         |
| ApiFactory.getInstance().getSession(conn, sessionId, version)          | Return a Session object, which is used to execute TADDM API methods.                                                                         |
| release()                                                             | This method is not supported, use the close() method                                                                                       |

### Examples

Connecting to the TADDM server involves establishing a connection to the server and logging in with a user account and password to establish a session. The following example illustrates how to establish a connection to the server:

```java
private ApiConnection conn_;  
try {  
    conn_ = ApiFactory.getInstance().getConnection(  
        "host.abcxyz.com", //host name  
        9530, //port number  
        null, //Location of jssecacerts.cert file for SSL  
        false); //true for SSL, false for non SSL  
} catch (Throwable th) {  
    th.printStackTrace();  }
```

Alternatively, you can establish an SSL connection, as illustrated by the following:

```java
private ApiConnection conn_;  
try {  
    conn_ = ApiFactory.getInstance().getConnection(  
        "host.abcxyz.com", //host name  
        9531, //port number  
        "c:\temp\jssecacerts.cert", //Location of jssecacerts.cert file for SSL  
        true); //true for SSL, false for non SSL  
} catch (Throwable th) {  
    th.printStackTrace();  }
```

The TADDM server uses ports 9530 and 9531 by default, though you can specify alternative ports by specifying appropriate values in $COLLATION_HOME/etc/collation.properties file.

When establishing an SSL connection, you must specify the location of the jssecacerts.cert file. Download this file from the TADDM server Java Web Start page, accessible at http://server_name:9430.

After the ApiConnection is established, log in to the server to establish a session, as illustrated by the following example:
String user = "smartoperator"; // login user name
String password = "foobar"; // user password
long version = 0; // CMDB version
ApiSession session_ = ApiFactory.getInstance().getSession(conn_, user, password, version);
CMDBApi api = session_.createCMDBApi();

CMDBApi is the remote reference to perform all API related operations on the TADDM server.

**Managing transactions:**

The transaction API has been deprecated.

The following methods will only log warning messages and transactions will not be started, committed, or rolled back:

beginTransaction()

beginTransaction(int timeout)

commitTransaction()

rollbackTransaction()

**Managing versions:**

Version methods manage TADDM database data versions. You can use the version methods to create named snapshots of the current TADDM database data, delete versions, and list the defined versions available.

*Table 19* describes the version methods you can use.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createEmptyVersion(name, description)</td>
<td>Create an empty version with no data.</td>
</tr>
<tr>
<td>createVersion(name, description)</td>
<td>Create a named snapshot of the current TADDM database data.</td>
</tr>
<tr>
<td>deleteVersion(versionID)</td>
<td>Delete the specified version from the TADDM database.</td>
</tr>
<tr>
<td>getAllVersions()</td>
<td>Return the names of all defined TADDM database data versions.</td>
</tr>
<tr>
<td>getVersion()</td>
<td>Return the TopologyVersion object of the current version of the displayed TADDM database data.</td>
</tr>
</tbody>
</table>

**Metadata:**

Metadata methods manage metadata in the Common Data Model. You can use the metadata methods to add, update, or remove extended attributes, and to set the associated values. You can also use the methods to return metadata information within the Common Data Model including the number, types, and names of all attributes in a model object.

*Table 20 on page 51* describes the metadata methods you can use.
### Table 20. Metadata methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defineExtendedAttributeMeta(udm)</td>
<td>Add or update class-wide extended attributes, or extended attributes for a specified class and account.</td>
</tr>
<tr>
<td>getAllMetaData()</td>
<td>Return all metadata. This method is equivalent to find(&quot;ObjectClass&quot;, ...) where a cache of metadata is used on the server for faster access.</td>
</tr>
<tr>
<td>getClassNames()</td>
<td>Return an array of model class short names, fully qualified name pairs.</td>
</tr>
<tr>
<td>getExtendedAttributeMeta(classname)</td>
<td>Retrieve extended attribute metadata for a class. The method retrieves class-wide and account-based extended attributes.</td>
</tr>
<tr>
<td>getExtendedAttributes(objGuid)</td>
<td>Retrieve extended attribute values for an object. The method retrieves class-wide and account-based extended attributes.</td>
</tr>
<tr>
<td>getMetaData(className)</td>
<td>Return the number, types, and names of all attributes in the model object. This method includes key/name rule, containment, relationship, and enumerated type information.</td>
</tr>
<tr>
<td>removeExtendedAttributeMeta(classname, acct)</td>
<td>Remove class-wide extended attributes or extended attributes for a specified class and account.</td>
</tr>
<tr>
<td>setExtendedAttributes(objGuid, attrNameVal)</td>
<td>Set the values of the extended attributes.</td>
</tr>
</tbody>
</table>

### Presentation:

Presentation methods determine affected systems and return topology information. You can use the presentation methods to determine affected business applications and services based on specific configuration items. You can also use the methods to return model object details, graphs, and topologies.

Table 21 describes the presentation methods you can use.

### Table 21. Presentation methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findImpactedBusinessApplications(objects)</td>
<td>Determine the business applications that are affected based on the specified array of Configuration Items.</td>
</tr>
<tr>
<td>findImpactedBusinessServices(objects)</td>
<td>Determine the business services that are affected based on the specified array of Configuration Items.</td>
</tr>
<tr>
<td>getDetailsPanel(ref)</td>
<td>Return the details panel for the specified object reference. An object reference is the combination of the object GUID and the version.</td>
</tr>
<tr>
<td>getGraphView(graphView)</td>
<td>Return a graph for the specified ViewDefiner that describes the graph view.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>getGraphViewImage(graphView)</td>
<td>Return an ImageStream object for the specified ViewDefiner that describes the graph view.</td>
</tr>
<tr>
<td>getTreeView(treeView)</td>
<td>Return a TopologyTreeModel object for the specified ViewDefiner that describes the tree view.</td>
</tr>
</tbody>
</table>

**Example Java code**

The following Example Java code illustrates how to retrieve the details panel for objects using the GUID.

**Note:**
- GUID is a unique identifier, used to identify objects while storing the objects in the database.
- ObjectRef is a simple data structure that holds the GUID and the version of the object you are interested in retrieving the details panel for.
- You can retrieve the GUID for the object you need by entering a command similar to the following command. This particular command gives you a list of all objects associated with ComputerSystem.

```
SELECT * FROM ComputerSystem
```

```
ModelObject mo[] = api.find(
    "SELECT * FROM ComputerSystem",
    1,
    null,
    null);
if (mo != null) {
    for (int i=0; i<mo.length; i++){
        Guid guid = mo[i].getGuid();
        System.out.println("Getting details panel for " + guid);
        DetailPanelModel model = api.getDetailsPanel( //guid and version
            new ObjectRef(guid,0)); //guid and version
        System.out.println("model is " + model);
    }
}
```

The following illustrates how to retrieve tree and graph views.

```
ViewDefiner viewDefiner = new ViewDefiner(
    ViewDefinerEnum.GRAPH_APPLICATION_PHYSICAL_INFRASTRUCTURE,
    VersionedObject.DYNAMIC);
TopologyGraphModel gv = api.getGraphView(viewDefiner);
```

The enumeration of graphs and trees are defined in the com.collation.proxy.api.presentation.common.ViewDefinerEnum class.

The following illustrates how to find impacted business services and applications:

```
// Find the objects for the impact analysis
ModelObject mo[] = api.find("SELECT * FROM ApacheServer", 1, null, null);
if (mo != null) {
    Application[] applications = api.findImpactedBusinessApplications(
        new Guid[] {mo[0].getGuid()});
}
```
if (applications != null) {
    for (int i=0; i<applications.length; i++) {
        System.out.println(applications[i].getDisplayName());
    }
}

BusinessSystem[] systems = api.findImpactedBusinessServices(
    new Guid[]{mo[0].getGuid()});
if (systems != null) {
    for (int i=0; i<systems.length; i++) {
        System.out.println(systems[i].getDisplayName());
    }
}

Security:

Security methods manage permissions, entitlements, and roles within the TADDM database. You can use the security methods to add and remove permissions, and determine the permissions and entitlements of specific users. You can also use the methods to determine the roles assigned to a user and determine whether a user has access to one or more runtime operations.

Table 22 describes the security methods you can use.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addAccess(user, resource, role, permissions)</td>
<td>Add permission for a specific object to a role.</td>
</tr>
<tr>
<td>addRuntimeAccess(user, role, permissions)</td>
<td>Add permission for one or more runtime operations to a role.</td>
</tr>
<tr>
<td>assignPersonInRoleToAccessCollection(user, role, guids, versionId)</td>
<td>Create an assignment (in potentially multiple versions) between a person in a role and a list of access collections.</td>
</tr>
<tr>
<td>deleteAccess(user, resource, role, permissions)</td>
<td>Delete a permission for a specific collection from a role.</td>
</tr>
<tr>
<td>deleteRuntimeAccess(user, role, permissions)</td>
<td>Delete permission for one or more runtime operations from a role.</td>
</tr>
<tr>
<td>getAccessDecisions(user, resources, permissions)</td>
<td>Determine whether the caller can access one or more objects with the specified permission.</td>
</tr>
<tr>
<td>getDataPermissions(user, resources)</td>
<td>Determine the data-level permissions that a user has for a set of objects.</td>
</tr>
<tr>
<td>getEntitlements(user, permissions)</td>
<td>Retrieve entitlements for the user. The entitlements are the objects that the user can access, based on the defined security policies.</td>
</tr>
<tr>
<td>getRoles(user)</td>
<td>Retrieve the roles assigned to a user.</td>
</tr>
<tr>
<td>getRuntimeAccess(user)</td>
<td>Retrieve permissions for runtime operations for a user.</td>
</tr>
<tr>
<td>getRuntimeAccessDecisions(user, permissions)</td>
<td>Determine if a user has access to one or more runtime operations.</td>
</tr>
</tbody>
</table>
Table 22. Security methods (continued)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>removePersonInRoleToAccessCollection(user,</td>
<td>Remove an assignment in potentially multiple versions based on the specified</td>
</tr>
<tr>
<td>role, guids, versionId)</td>
<td>person, role, and list of access collections to which the person-in-role is</td>
</tr>
<tr>
<td></td>
<td>assigned.</td>
</tr>
<tr>
<td>addAccess(user, resource, accessDefinitions)</td>
<td>Add one or more objects (each with a set of permissions) to a role, as</td>
</tr>
<tr>
<td></td>
<td>specified by each AccessDefinition object.</td>
</tr>
<tr>
<td>addDataPermissionToRole(role, permission)</td>
<td>Add a data permission to a role wherever the role exists in the stored</td>
</tr>
<tr>
<td></td>
<td>policies.</td>
</tr>
<tr>
<td>addRuntimePermissionToRole(role, permission)</td>
<td>Add a runtime permission to a role.</td>
</tr>
<tr>
<td>deletePermission(permission)</td>
<td>Remove a permission wherever the permission exists in the stored policies.</td>
</tr>
<tr>
<td>deletePermissionFromRole(role, permission)</td>
<td>Remove a permission from a role wherever the role exists in the stored</td>
</tr>
<tr>
<td></td>
<td>policies.</td>
</tr>
<tr>
<td>deleteRole(role)</td>
<td>Remove a role wherever the role exists in the stored policies.</td>
</tr>
<tr>
<td>getEntitlementsForRole(user, role)</td>
<td>Retrieve entitlements for the user in a specified role.</td>
</tr>
</tbody>
</table>

Managing application templates:

Application template methods enable the creation, modification, deletion, and retrieval of application templates and rules.

Application templates specify the MQL rules that are periodically applied to the TADDM database to define business applications or collections. Each template specifies one or more rules with the following attributes:

**MQLRuleName**
- The name of the MQL rule. This attribute is required.

**FunctionalGroupName**
- The object the functional group contains that the MQL query returns. This attribute is required for any rule that defines a business application. It is not used for collections that rules define.

**MQLQuery**
- The MQL query to run. The objects the query returns are added to the business application or collection.

Table 23 on page 55 describes the application template methods that you can use.
Table 23. Application template methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createAppTemplate(String name, String type, boolean removeNonMembers, MQLRule[] operators)</td>
<td>Creates a template with the specified rules. The following parameters are available:</td>
</tr>
<tr>
<td>name</td>
<td>The name of the application template to create.</td>
</tr>
<tr>
<td>type</td>
<td>The type of template to create (application or collection or service). Valid values are “Application” for Business Application or “Collection” for Collections or “Service” for Business Service.</td>
</tr>
<tr>
<td>removeNonMembers</td>
<td>A Boolean value that specifies whether existing objects in the business application or collection can be removed if they no longer match the template rules.</td>
</tr>
<tr>
<td>MQLRule</td>
<td>An array that contains one or more rules.</td>
</tr>
<tr>
<td>updateAppTemplate(String name, String type, boolean removeNonMembers, MQLRule[] operators)</td>
<td>Updates a template with the specified rules. The following parameters are available:</td>
</tr>
<tr>
<td>name</td>
<td>The name of the application template to update.</td>
</tr>
<tr>
<td>type</td>
<td>The type of template to update (application or collection or service). Valid values are “Application” for Business Application or “Collection” for Collections or “Service” for Business Service.</td>
</tr>
<tr>
<td>removeNonMembers</td>
<td>A Boolean value that specifies whether existing objects in the business application or collection can be removed if they no longer match the template rules.</td>
</tr>
<tr>
<td>MQLRule</td>
<td>An array that contains one or more rules.</td>
</tr>
<tr>
<td>getAllAppTemplates()</td>
<td>Retrieves all application templates.</td>
</tr>
<tr>
<td>getAppTemplate(String name, int type)</td>
<td>Retrieves the application template for specified name and type.</td>
</tr>
<tr>
<td>removeAppTemplate(String name, int type)</td>
<td>Removes the application template for specified name and type.</td>
</tr>
<tr>
<td>removeMQLRule(String name)</td>
<td>Removes an MQL rule. A rule can be removed only if it is not associated with any application templates.</td>
</tr>
</tbody>
</table>
Creating a business application template

To create an application template for a business application, complete the following steps:
1. Create a business application and set the name of the new business application to the name passed on the command line, for example, TADDM - Production. Get the GUID returned.
2. Set the name of the application template using the following format:
   \[business\_app\_GUID:business\_app\_name\]
   For example,
   AA0A20EE5BBD336481279CA664FB380A:TADDM - Production
3. Prefix the rule names with the GUID of the business application, but ensure that you do not include a colon in the rule name, for example
   AA0A20EE5BBD336481279CA664FB380Adatabase

Example: Creating a business application template

The following example creates a business application template, the MQL rules, and the associated business application:

```java
# create Business Application with name "TADDM - Production"
BAname = "TADDM - Production"

# get guid of Business Application
myBA = ModelFactory.newInstance(Class.forName("com.collation.platform.model.topology.app.Application"))
myBA.setName(BAname)
appGuid = api.update(myBA, None)

MQLRuleClass = Class.forName("com.collation.platform.model.apptemplate.MQLRule")
rules = [ModelObjectFactory.newInstance(MQLRuleClass)]

# create required by TADDM MQLRule name like AA0A20EE5BBD336481279CA664FB380Adatabase
rulename = "database"
RuleName = str(appGuid) + rulename
asQuery = "select * from AppServer"
rules[0].setMQLRuleName(RuleName)
rules[0].setFunctionalGroupName("App Servers")
rules[0].setMQLQuery(asQuery)

# create required by TADDM AppTemplate name like
appTemplateName = str(appGuid) + ":" + BAname
appTemplate = api.createAppTemplate(appTemplateName, "APPLICATION", True, jarray.array(rules, MQLRuleClass));
```

Example: Listing business applications

The following example lists business applications:

```java
query = "select * from Application"
data = api.executeQuery(query, None, None)
while (data.next()):
    print data.getXML(4)
```

Example: Removing a business application template

The following example removes an application template, the MQL rules, and the associated business application.

```java
# Get application template
appTemplate = api.getAppTemplate(nameBA, 0)
```
# Get rules of the application template
rules = appTemplate.getMQLRules()

# Remove application template -> removes just AppTemplate object
api.removeAppTemplate(appTemplate.getAppTemplateName(), appTemplate.getAppTemplateType())

# Remove all rules
for rule in rules:
    rule = api.find(rule.getGuid(), 1, None)
    api.removeMQLRule(rule.getMQLRuleName())

# Remove business application
businessApp = api.find("Select * from Application where name =='" +
appTemplate.getAppTemplateName()+ '''", False, None, None)
api.delete(businessApp, None)

Using the SOAP API

The SOAP API exposes elements of the TADDM API as a web service.

Using the SOAP API, you can develop applications across a range of development environments and operating systems supporting integration with management applications including ITSM Process Managers.

The SOAP API provides control over the discovery process and aspects of the Common Data Model including access to the resulting model data. The SOAP API delegates requests to the Java API. The Java API can create applications that add, update, and delete model objects. You can query model objects by class name or object ID number.

You can use the SOAP API control to create applications that add, update, and delete model objects. SOAP can query model objects by class name or object ID number. You can also use the interface to examine the change history and manage versions.

Request summary

The SOAP API offers access to the TADDM application maps, including the discovered applications, their components, and configurations.

The SOAP API can be summarized using the following categories, which are explained in detail in their appropriate sections:

- Session requests
- Discovery requests
- Managing the model and metadata
- Find requests
- Change history requests
- Managing versions

**Session requests:**

Session requests enable you to manage sessions with the TADDM server.

You can use the session requests to login and logout on the TADDM server. [Table 24 on page 58](#) describes the session requests you can use.
Table 24. Session requests

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>loginRequest</td>
<td>loginResponse</td>
</tr>
<tr>
<td>(Log in to the TADDM server)</td>
<td>user</td>
<td>The user name registered with the TADDM server</td>
</tr>
<tr>
<td></td>
<td>password</td>
<td>The password associated with the user</td>
</tr>
<tr>
<td></td>
<td>host</td>
<td>The name of the host, either as a name, or as an IP address (using dot notation)</td>
</tr>
<tr>
<td></td>
<td>port</td>
<td>The port number of the server</td>
</tr>
<tr>
<td>logout</td>
<td>logoutRequest</td>
<td>logoutResponse</td>
</tr>
<tr>
<td>(Log out from the server)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

The following example shows a login XML request:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:login soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://localhost">
      <ns1:arg0 xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">smartoperator</ns1:arg0>
      <ns1:arg1 xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">foobar</ns1:arg1>
      <ns1:arg2 xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">localhost</ns1:arg2>
    </ns1:login>
  </soapenv:Body>
</soapenv:Envelope>
```

The following example shows the login XML response:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:loginResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://localhost">
      <loginReturn xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">1149902064172</loginReturn>
    </ns1:loginResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Discovery requests:

Discovery methods enable you to manage discovery runs.
You can use the discovery requests to start and stop discoveries, get the status of a discovery, and clear all discovery elements from the topology. Table 25 describes the discovery requests you can use.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortDiscovery</td>
<td>abortDiscoveryRequest</td>
<td>abortDiscoveryResponse</td>
</tr>
<tr>
<td>(Abort the currently running discovery)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clearTopology</td>
<td>clearTopologyRequest</td>
<td>clearTopologyResponse</td>
</tr>
<tr>
<td>(Clear all discovery elements and relationships from the topology)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getStatus</td>
<td>getStatusRequest</td>
<td>getStatusResponse</td>
</tr>
<tr>
<td>(Get the current discovery run status)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rebuildTopology</td>
<td>rebuildTopologyRequest</td>
<td>rebuildTopologyResponse</td>
</tr>
<tr>
<td>(Rebuild the topology, including dependencies and relationships)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>startDiscovery</td>
<td>startDiscoveryRequest</td>
<td>startDiscoveryResponse</td>
</tr>
<tr>
<td>(Start a discovery using the specified scope)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Managing the model and metadata:**

Model and metadata requests manage objects and query metadata in the Common Data Model. You can use the model requests to insert, import, and export objects in the Common Data Model. You can use the metadata request to get all class names in the model that can be used in the query language.

Table 26 on page 60 describes the model and metadata requests you can use.
### Table 26. Model and metadata requests

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>exportData</td>
<td>exportDataRequest</td>
<td>exportDataResponse</td>
</tr>
<tr>
<td>(Export all top level objects in the TADDM database to a specified directory in XML format)</td>
<td>directoryToWriteTo&lt;br&gt;The name of the directory to which the data is to be exported&lt;br&gt;maxfilesize&lt;br&gt;The maximum file size to export&lt;br&gt;mssGuid&lt;br&gt;The GUID of the Management Software System (MSS)</td>
<td></td>
</tr>
<tr>
<td>exportDataUsingMssName</td>
<td>exportDataUsingMssNameRequest</td>
<td>exportDataUsingMssNameResponse</td>
</tr>
<tr>
<td>(Export all top-level objects in the TADDM database to a directory, in XML format, specifying the MSS using a name)</td>
<td>directoryToWriteTo&lt;br&gt;The name of the directory to which the data is to be exported&lt;br&gt;maxfilesize&lt;br&gt;The maximum file size to export&lt;br&gt;mssGuid&lt;br&gt;The GUID of the Management Software System (MSS)</td>
<td></td>
</tr>
<tr>
<td>getClassNames</td>
<td>getClassNamesRequest</td>
<td>getClassNamesResponse</td>
</tr>
<tr>
<td>(Get all class names in the model that can be used in the query language)</td>
<td>getClassNamesReturn: Array of model class short name/fully qualified name pairs</td>
<td></td>
</tr>
<tr>
<td>importData</td>
<td>importDataRequest</td>
<td>importDataResponse</td>
</tr>
<tr>
<td>(Convert XML data from the specified source into model objects and update the TADDM database)</td>
<td>source&lt;br&gt;The source from which to import the data&lt;br&gt;rebuildTopo&lt;br&gt;A Boolean to rebuild the topology&lt;br&gt;mssGuid&lt;br&gt;The GUID of the Management Software System (MSS)</td>
<td></td>
</tr>
</tbody>
</table>
Table 26. Model and metadata requests (continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>importDataUsingMssName</td>
<td>importDataUsingMssNameRequest</td>
<td>importDataUsingMssNameResponse</td>
</tr>
<tr>
<td>(Convert XML data from the specified source into model objects and update the TADDM database, specifying Management Software System by name)</td>
<td>source The source from which to import the data</td>
<td>rebuildTopo A Boolean to rebuild the topology</td>
</tr>
<tr>
<td></td>
<td>mssName The name of the Management Software System (MSS)</td>
<td>importDataUsingMssNameRequest</td>
</tr>
<tr>
<td>insert</td>
<td>insertRequest</td>
<td>insertResponse</td>
</tr>
<tr>
<td>(Insert or update the model object, specified in XML format, in the TADDM database)</td>
<td>xml The model object in XML format</td>
<td>insertResponse</td>
</tr>
<tr>
<td></td>
<td>mssGuid The GUID of the Management Software System (MSS)</td>
<td>insertResponse</td>
</tr>
<tr>
<td>insertUsingMssName</td>
<td>insertUsingMssNameRequest</td>
<td>insertUsingMssNameResponse</td>
</tr>
<tr>
<td>(Insert the model object, specified in XML format, specifying the Management Software System by name)</td>
<td>xml The model object in XML format</td>
<td>insertUsingMssNameResponse</td>
</tr>
<tr>
<td></td>
<td>mssName The name of the Management Software System (MSS)</td>
<td>insertUsingMssNameRequest</td>
</tr>
</tbody>
</table>

Example

The following example shows a getClassNames XML request:

```xml
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  <ns1:getClassNames soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://localhost"/>
</soapenv:Body></soapenv:Envelope>
```

The following example shows the getClassNames XML response:

GETCLASSNAME (response):

```xml
<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
</soapenv:Body></soapenv:Envelope>
```
Find requests:

Find requests enable you to access objects in the Common Data Model.

You can use the find requests to return model objects matching a specific criteria or return information about specific managed elements. You can also use the requests to return objects that have changed during a specified period of time. Table 27 describes the find operations you can perform.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td>findRequest</td>
<td>findResponse</td>
</tr>
<tr>
<td>(Execute a query for a configuration item specified using the Model Query Language)</td>
<td>query The query string</td>
<td>findResponse: XML representation of the query results</td>
</tr>
<tr>
<td></td>
<td>depth The level of the result tree to construct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indent The indentation to use for the resulting XML file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mssGuid The GUID of the Management Software System (MSS)</td>
<td></td>
</tr>
<tr>
<td>findBasedOnChange</td>
<td>findBasedOnChangeRequest</td>
<td>findBasedOnChangeResponse</td>
</tr>
<tr>
<td>(Find objects that changed in the specified period for a given change type)</td>
<td>root The model object to serve as the root for the resulting XML output.</td>
<td>findBasedOnChangeReturn: XML representation of the query results</td>
</tr>
<tr>
<td></td>
<td>depth The level of the result tree to construct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indent The indentation to use for the resulting XML file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>start The start time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end The end time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>changeType The type of change.</td>
<td></td>
</tr>
</tbody>
</table>
Table 27. Find requests (continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>findUsingGuid</td>
<td>findUsingGuidRequest</td>
<td>findUsingGuidResponse</td>
</tr>
<tr>
<td>(Find using the GUID of the model object)</td>
<td>guid Guid against which to execute the find</td>
<td>findUsingGuidReturn: XML representation of the query results</td>
</tr>
<tr>
<td></td>
<td>depth Level of the result tree to construct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indent Indentation to use for the resulting XML file</td>
<td></td>
</tr>
<tr>
<td>findUsingMssName</td>
<td>findUsingMssNameRequest</td>
<td>findUsingMssNameResponse</td>
</tr>
<tr>
<td>(Find using the Model Query Language, specifying the Management Software System by name)</td>
<td>query Query string</td>
<td>findUsingMssNameReturn: XML representation of the query results</td>
</tr>
<tr>
<td></td>
<td>depth Level of the result tree to construct</td>
<td></td>
</tr>
<tr>
<td></td>
<td>indent Indentation to use for the resulting XML file</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mssName Name of the Management Software System (MSS)</td>
<td></td>
</tr>
</tbody>
</table>

Example

The following example shows a find XML request:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:find soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://localhost">
      <ns1:arg0 xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">ComputerSystem</ns1:arg0>
      <ns1:arg1 href="#id0"/>
      <ns1:arg2 href="#id1"/>
      <ns1:arg3 xsi:nil="true"/>
    </ns1:find>
  </soapenv:Body>
</soapenv:Envelope>
```

The following example shows the find XML response:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
      <findReturn xsi:type="soapenc:string" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <ref id="id1" soapenc:root="0" soapenc:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="soapenc:int" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">4</ref>
        <ref id="id0" soapenc:root="0" soapenc:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xsi:type="soapenc:int" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">2</ref>
      </findReturn>
    </ns1:findResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
Change history requests:

Change history requests enable you to determine the change history within the Common Data Model.

You can use the change history requests to retrieve the change history for managed elements within the Common Data Model. Table 28 describes the change history requests you can use.

Table 28. Change history requests

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>getChangeHistory</td>
<td>getChangeHistoryRequest</td>
<td>getChangeHistoryResponse: XML representation of a list of ChangeHistory objects</td>
</tr>
<tr>
<td>(Get the change history for the start and end period using the specified GUID)</td>
<td>guid: The GUID of the object for which the change history is required</td>
<td>getChangeHistoryReturn: XML representation of a list of ChangeHistory objects</td>
</tr>
<tr>
<td></td>
<td>start: The start time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end: The end time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT</td>
<td></td>
</tr>
</tbody>
</table>
Table 28. Change history requests (continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>getChangeHistory</td>
<td>getChangeHistoryRequest1</td>
<td>getChangeHistoryResponse1</td>
</tr>
<tr>
<td>(Get the change</td>
<td>guid</td>
<td>getChangeHistoryReturn:</td>
</tr>
<tr>
<td>history for the</td>
<td>The list of comma-separated</td>
<td>XML representation of a list of ChangeHistory objects</td>
</tr>
<tr>
<td>start and end</td>
<td>GUIDs of the objects for</td>
<td></td>
</tr>
<tr>
<td>period for</td>
<td>which the change history is</td>
<td></td>
</tr>
<tr>
<td>multiple GUIDs)</td>
<td>required</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>The start time for the</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td>change period, specified in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>milliseconds since January</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1, 1970, 00:00:00 GMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>end</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The end time for the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>change period, specified in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>milliseconds since January</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1, 1970, 00:00:00 GMT</td>
<td></td>
</tr>
</tbody>
</table>

Managing versions:

Version requests manage TADDM database data versions. You can use the version requests to create named snapshots of the current TADDM database data, delete versions, and list the defined versions available.

Table 29 describes the version requests you can use.

Table 29. Version requests

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>createVersion</td>
<td>createVersionRequest</td>
<td>createVersionResponse</td>
</tr>
<tr>
<td>(Create a named</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>snapshot of the</td>
<td>The name of the version</td>
<td></td>
</tr>
<tr>
<td>current TADDM</td>
<td>description</td>
<td></td>
</tr>
<tr>
<td>database data)</td>
<td>A description of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new version</td>
<td></td>
</tr>
<tr>
<td>createEmptyVersion</td>
<td>createEmptyVersionRequest</td>
<td>createEmptyVersionResponse</td>
</tr>
<tr>
<td>(Create an empty</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>version, with no</td>
<td>The name of the version</td>
<td></td>
</tr>
<tr>
<td>data, in the</td>
<td>description</td>
<td></td>
</tr>
<tr>
<td>TADDM database)</td>
<td>A description of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>new version</td>
<td></td>
</tr>
<tr>
<td>deleteVersion</td>
<td>deleteVersionRequest</td>
<td>deleteVersionResponse</td>
</tr>
<tr>
<td>(Delete a version</td>
<td>versionID</td>
<td></td>
</tr>
<tr>
<td>from the TADDM</td>
<td>The identifier of the</td>
<td></td>
</tr>
<tr>
<td>database)</td>
<td>version</td>
<td></td>
</tr>
</tbody>
</table>
Table 29. Version requests (continued)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteVersionUsingName</td>
<td>deleteVersionUsing NameRequest</td>
<td>deleteVersionUsing NameResponse</td>
</tr>
<tr>
<td>(Delete a version, identified</td>
<td>versionName</td>
<td></td>
</tr>
<tr>
<td>by name, from the TADDM database)</td>
<td>The name of the version</td>
<td></td>
</tr>
<tr>
<td>getAllVersions</td>
<td>getAllVersionsRequest</td>
<td>getAllVersionsResponse</td>
</tr>
<tr>
<td>(Get the names of all defined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TADDM database data versions)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Developing applications using the REST API

You can use the TADDM REST API to develop applications that access selected TADDM resources using HTTP and REST principles.

REST API overview

The REST API exposes a subset of the Java API functions to clients and Web browsers using HTTP. Using the REST resources, you can develop applications for any operating system and language that supports HTTP calls.

The REST resources expose TADDM functions you can use to query model objects by class name, by globally unique identifier (GUID), or with Model Query Language (MQL) queries. You can also create, delete, and update model objects, as well as manage the TADDM discovery process. All of these functions use standard HTTP interfaces and support either JSON or XML format for input and output data.

The REST server components are installed in the $COLLATION_HOME/deploy-tomcat directory on the TADDM server and start automatically when the TADDM server starts. The REST services are available using the same TCP/IP ports used by the TADDM administrative Web interface. (The default ports are 9430 for HTTP and 9431 for HTTPS.)

The REST API uses HTTP Basic authentication for transmitting the user ID and password using MIME Base64 encoding. Because each request is stateless, each call to the REST API must include the HTTP authorization header. For secure connections, use an HTTPS connection.

Parameters for REST calls are specified using standard query string notation:
http://resource_url?parameter=value&parameter=value...

If you specify a parameter value that is not valid, the TADDM server disregards the value and uses the default value, if one can be determined. (For example, if you specify fetchSize=-2, the server uses a fetchSize value of 1.) If no default value can be determined, the request fails.

Making REST calls with a Web browser

You can make many REST API calls by entering the appropriate URLs in a Web browser.
Before you begin

To access the REST interfaces securely using an HTTPS connection, you must first configure your browser to accept Transport Layer Security (TLS) 1.0 connections.

About this task

The REST API uses several HTTP methods to perform various actions on REST resources. Any REST API call that uses the HTTP GET method can be submitted using a Web browser such as Microsoft Internet Explorer or Mozilla Firefox.

Procedure

To access a REST call with a browser, complete the following steps:

Enter the appropriate URL using either HTTP or HTTPS.

- This example submits a query specified with Model Query Language using HTTP:
  
  http://yourhost.com:9430/rest/model/MQLQuery?query=select%20name%20from%20ComputerSystem%20where%20signature%20starts-with%20'M&Y'&feed=xml&fetchSize=100&position=1

- This example shows a ComputerSystem query submitted using HTTPS:
  
  https://yourhost.com:9431/rest/model/ComputerSystem?depth=1

The first time you access the TADDM REST API using a browser, a login page prompts you for a valid TADDM user ID and password.

Making REST calls in a Java application

You can use standard Java methods to access the TADDM REST API.

Before you begin

To access the REST interfaces securely using an HTTPS connection, you must first copy the jssecacerts.cert security certificate to the client system. This file is located in the /etc directory on the TADDM server.

Procedure

The following example shows how to access the REST API from a Java program.

To access the REST API from a Java program, use the standard Java methods for HTTP communication. This example accesses the REST API using a secure HTTPS connection:

```java
HostnameVerifier hv = new HostnameVerifier() {
    public boolean verify(String urlHostName, SSLSession session) {
        System.out.println("Warning: URL Host: "+urlHostName+" vs. "
        +session.getPeerHost());
        return true;
    }
};

// set this property to the location of the cert file
System.setProperty("javax.net.ssl.trustStore", "jssecacerts.cert");

HttpsURLConnection.setDefaultHostnameVerifier(hv);
    "Repository?depth=1&feed=json");
```
Parsing REST query results
To parse REST query results in a Java application, you can use standard XPath or JXPath methods.

Before you begin
Make sure you have access to an XPath library for parsing XML data, or a JXPath library for parsing JSON data. XPath support is included as part of Java versions 5.0 and later; JXPath support is included with the TADDM SDK.

Procedure
The following example shows how to use these functions to parse the output from TADDM REST calls.

You can then use these functions to parse the output from TADDM REST calls. The following example shows how you might return the serviceName of each of the installedServices for an operating system, using JXPath to parse JSON output data.

Note: The JSONArray class is part of the json-simple package, which is not included in the TADDM SDK. To download this package, go to http://code.google.com/p/json-simple/

```
//queryResult contains the results from a TADDM Query
JSONArray arrayObject = (JSONArray) JSONValue.parse(queryResult);
            JXPathContext context2 = JXPathContext.newContext(arrayObject);
            Iterator names = context2.iterate("//installedServices/serviceName");
            while(names.hasNext()) {
                String serviceName = (String) names.next();
                System.out.println("service name is: " + serviceName);
            }
```

Debugging REST applications
If your application is encountering errors while accessing the REST API, there are several techniques you can use to determine the nature of the problem.
About this task

The REST API uses several mechanisms to indicate the results of REST calls and errors that occur during processing. Use these methods to debug a REST application.

Procedure

Use the following methods to debug a REST application.

- Check the HTTP response code. Commonly used response codes include the following:
  - 200 The request was successful.
  - 400 The input data was not valid.
  - 409 The server encountered a conflict such as an attempt to add an object that already exists.
  - 500 A server error occurred.
- Check the response message in the HTTP header for additional information.
- Check the log files for any relevant messages. Messages might appear in the following files:
  - $COLLATION_HOME/log/tomcat.log
  - $COLLATION_HOME/log/services/ApiServer.log

Querying model objects using the REST API

You can use the REST API to query model objects using either of two methods.

Procedure

To use the REST API to query model objects, complete one of the following two methods:

- To query model objects by specifying the model object class, use the model object class resource. You can use this resource to query information about model objects of a particular class, optionally including specified attribute values. This resource provides a simple way to query objects of a particular class. This example queries the fifth ComputerSystem object whose OSRunning attribute is set to Linux:
  ```
  ```

- To query model objects using Model Query Language (MQL), use the MQL query service resource. This resource supports any query that can be expressed using MQL and is more flexible than the model object class resource. This example queries model object data using the MQL query select displayName,OSRunning.OSName from ComputerSystem.
  ```
  http://example.com:9430/rest/model/MQLQuery?query=select%20displayName,OSRunning.OSName%20from%20ComputerSystem&position=2&fetchSize=2&feed=xml&depth=2&position=4
  ```

Adding model objects using the REST API

You can use the REST API to add a new model object using either of two methods.
About this task

The model object update service resource supports creation of new model objects using the HTTP POST or PUT method, depending on whether you want to allow modification of an existing object.

In either case, you must first describe the new object data using either JSON or XML format; the server automatically detects which format is used. If you need to specify a complex model object, it can be useful to first query the class metadata using the model object class metadata service. The results of this query will provide the correct attribute names for the object, which you can then use to specify the new data in XML or JSON format.

In some situations, you might need to add a model object that includes another new model object as an attribute, with the parent attribute required on the child object. This requires that you first determine the GUID of the parent object so you can then set the parent attribute of the child object. You can accomplish this in either of two ways:

- Create the parent object first, omitting the child object, which enables you to determine the GUID of the parent. You can then create the child object, specifying the GUID of the parent object.

- Create both objects with a single request. To use this method, you must set the GUID of the parent object to a value that is unique within the JSON or XML document, and specify that same value on the parent attribute of the child object. This JSON example uses the ID cs1 as the GUID of the parent object:

  ```json
  [{"signature":"JsonRestExample1","class":"LinuxUnitaryComputerSystem","numCPUs":2,"guid":"cs1","OSRunning":{"_class":"Linux","parent":"cs1","name":"Linux","description":"Created by sample code"}]
  ```

  For more examples, refer to the sample programs in the $COLLATION_HOME/sdk/examples/rest directory.

Procedure

Use one of the following two methods:

- Use the model object update service and the HTTP POST method, passing the new object data in the body of the request. This method succeeds only if the specified object does not already exist. If the object already exists, the request fails. Use this method if you want to create a new object but do not want to make any changes to existing objects.

- Use the model object update service and the HTTP PUT method, passing the new object data in the body of the request. This method creates the specified object if it does not already exist; if the object already exists, it is modified with the new data. Use this method if you want to make sure the specified object is in the TADDM database, regardless of whether it already existed.

Updating model objects using the REST API

You can use the REST API to update an existing model object using either of two methods.

About this task

You can use either a model object resource or the model object update service resource to update an existing model object, depending on whether you want to allow creation of new objects.
In either case, you must first describe the new object data using either JSON or XML format; the server automatically detects which format is used. If you need to specify a complex model object, it can be useful to first query the class metadata using the model object class metadata service. The results of this query will provide the correct attribute names for the object, which you can then use to specify the new data in XML or JSON format.

**Procedure**

To use the REST API to update existing model objects, complete one of the following two methods:

- Use the model object resource representing the existing object and the HTTP PUT method, passing the new object data in the body of the request. This resource is available only if the specified object already exists; if the object does not exist, the request fails. Use this method if you want to modify an existing object but do not want to add the object if it does not exist.

- Use the model object update service and the HTTP PUT method, passing the new object data in the body of the request. This method updates the object with the new data; if the object does not already exist, this method creates it. Use this method if you want to make sure an object with the specified object data exists in the TADDM database, regardless of whether it already existed.

**Deleting model objects using the REST API**

You can use the REST API to delete a model object using either of two methods.

**About this task**

Only a single object can be deleted in a single request. If the specified object does not exist, no error is returned.

**Procedure**

To delete a model object, complete one of the following two methods:

- To delete an object by specifying its GUID, use the corresponding model object resource, specifying the GUID of the object to delete as part of the URL and using the HTTP DELETE method. If you use this method, no data is required in the body of the HTTP request. This example, submitted using the HTTP DELETE method, deletes an object using the model object resource:

  http://example.com:9430/rest/model/ModelObject/1D646C44FDEB3857B40B98BD9C0F407?msgGuid=CF5EBF574E7F382289B3F35F5B5776628

- Use the model object update service with the delete parameter and the HTTP POST method, specifying the object to delete in the body of the HTTP request. Only the GUID of the object to delete is required in the input data, although the entire object can be specified. This example, submitted using the HTTP POST method, deletes the object specified in the body of the request:

  http://example.com:9430/rest/model/ModelObject?delete=true

**Managing discoveries using the REST API**

You can use the REST API to start discovery and to manage discoveries, discovery profiles, and discovery scopes.

**Procedure**

To use the REST API, complete one or more of the following steps:
To start discovery, use the discovery service resource, specifying a name for the discovery run. You can use this resource to start discovery with or without a profile. For example, this REST call starts discovery using the Level 3 Discovery profile, with the scope limited to one IP address.

http://example.com:9430/rest/discovery/start/TestRun2?profile=Level%203%20Discovery&scope=192.168.100.101

It is now possible to run parallel discoveries using the REST API.

To check the current discovery status, use the discovery status resource, specifying either XML or JSON format for the output data. This example checks discovery status using JSON format:

http://example.com/rest/discovery/status?feed=json

To retrieve a list of defined discovery profiles, use the discovery profile service resource, specifying either XML or JSON format for the output data. This example lists discovery profiles using XML format:

http://example.com/rest/discovery/profiles?feed=xml

To retrieve details of a defined discovery profile, use the discovery profile resource, specifying either XML or JSON format for the output data. This retrieves details of the Level 3 Discovery profile using JSON format:

http://example.com/rest/discovery/profile/Level%203%20Discovery?feed=json

To retrieve a list of defined discovery scopes, use the discovery scope service resource, specifying either XML or JSON format for the output data. This example lists discovery scopes using XML format:

http://example.com/rest/discovery/scopes?feed=xml

To retrieve details of a defined discovery scope, use the discovery scope resource, specifying either XML or JSON format for the output data. This retrieves details of the scope1 scope using JSON format:

http://example.com/rest/discovery/scope/scope1?feed=json

REST resource reference

The REST API exposes resources you can use to query, create, update, and delete model objects, and to manage discoveries.

Model object class:

The model object class resource represents a class of model objects defined by the Common Data Model.

Description

Use this resource to retrieve information about model objects by specifying a model object class, optionally including attribute values. This type of request provides a subset of the information available through MQL queries.

Use the HTTP GET method to send an MQL query request.

URL

scheme://hostname:port/rest/model/model_object_class

where:

scheme
The scheme of the URL (either HTTP: or HTTPS:).
hostname
The TCP/IP hostname or numeric IP address of the TADDM server.

port
The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).

model_object_class
The model object class name. Specify either the short name (such as ComputerSystem) or the fully qualified name (such as com.collation.platform.model.topology.sys.ComputerSystem).

HTTP methods
GET Queries model objects.

Parameters
cols=value
A comma-separated list of the column names for which you want data to be returned. The default is to return data from all columns.

depth=value
The depth of the query. The default value is 1.

Note: A query with a depth greater than 1 can return a large result set, causing low-memory conditions on the TADDM server. To avoid this problem, specify fetchSize=1 and use consecutive queries to scroll through the data one position at a time. Refer to the sample programs in the $COLLATION_HOME/sdk/examples/rest directory to see examples of how to use this technique.

feed={json|xsl}
The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

fetchSize=value
The maximum number of objects to return from the result set. The default value is 1.

longClassName={true|false}
Specifies whether all model object class names in the output must be specified using the fully qualified form (for example, com.collation.platform.model.topology.sys.ComputerSystem). Specify true or false. This option is valid only for JSON output. The default value is false.

mssGuid=value
The GUID value of the management software system (MSS) associated with the object. This parameter is optional.

position=value
The starting position in the result set for the objects you want returned from the query. The default value is 1 (the first object in the result set). If you specify a position that is greater than the total number of objects in the result set, no objects are returned.

attribute_name=attribute_value
An optional attribute name and value. Use this option to limit the query
output to objects matching the specified attribute value. If you specify more
than one attribute, only objects matching all of the specified attribute values
are returned.

Returns

If the query is successful, the server returns the HTTP return code 200, and the
query result data in either JSON or XML format (as specified by the feed
parameter or the HTTP Accept header). If the query returns no data, the result set
is an empty JSON array or XML document, depending on the feed type.

The TADDMQueryComplete pragma header of the returned data indicates whether
all available query results have been returned; true indicates that all results have
been returned, and false indicates that more query results are available. You can
control which results are returned by adjusting the values of the optional position
and fetchSize parameters.

Example

This example queries the fifth ComputerSystem object whose OSRunning attribute
is set to linux:

http://example.com:9430/rest/model/ComputerSystem?depth=2&feed=xml&OSRun-
n ing.OSName=Linux&position=5

MQL query service:

The MQL query service resource retrieves model object data based on queries
written in the Model Query Language (MQL).

Description

Use this resource to retrieve model object data using queries written in MQL. The
MQL query service can provide more detailed information than is available from
the Model Object Class resource.

URL

scheme://hostname:port/rest/model/MQLQuery

where:
scheme
   The scheme of the URL (either HTTP: or HTTPS:).
hostname
   The TCP/IP hostname or numeric IP address of the TADDM server.
port
   The TCP/IP port on the TADDM server for the type of connection you are
   using (9430 for HTTP, or 9431 for HTTPS).

HTTP methods

GET   Queries model objects.

Parameters

depth=value
   The depth of the query. The default value is 1.
Note: A query with a depth greater than 1 can return a large result set, causing low-memory conditions on the TADDM server. To avoid this problem, specify fetchSize=1 and use consecutive queries to scroll through the data one position at a time. Refer to the sample programs in the $COLLATION_HOME/sdk/examples/rest directory to see examples of how to use this technique.

feed={json|xml}

The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

fetchSize=value

The maximum number of objects to return from the result set. The default value is 1.

longClassName={true|false}

Specifies whether all model object class names in the output must be specified using the fully qualified form (for example, com.collation.platform.model.topology.sys.ComputerSystem). Specify true or false. This option is valid only for JSON output. The default value is false.

mssGuid=value

The GUID value of the management software system (MSS) associated with the object. This parameter is optional.

position=value

The starting position in the result set for the objects you want returned from the query. The default value is 1 (the first object in the result set). If you specify a position that is greater than the total number of objects in the result set, no objects are returned.

query=value

The query string, written in MQL notation. This parameter is required.

Note: Model object queries can return large amounts of data. To avoid memory and performance problems, select only the columns you need.

Returns

If the query is successful, the server returns the HTTP return code 200, and the query result data in either JSON or XML format (as specified by the feed parameter or the HTTP Accept header). If the query returns no data, the result set is an empty JSON array or XML document, depending on the feed type.

The TADDMQueryComplete pragma header of the returned data indicates whether all available query results have been returned; true indicates that all results have been returned, and false indicates that more query results are available. You can control which results are returned by adjusting the values of the optional position and fetchSize parameters.

Example

This example queries model object data using the MQL query select displayName,OSRunning.OSName from ComputerSystem.
Model object:

A model object resource represents a specific model object instance that exists in the TADDM database, identified by GUID.

Description

Use this type of resource to query, update, or delete a single model object instance identified by its globally unique identifier (GUID).

URL

```
scheme://hostname:port/rest/model/ModelObject/guid
```

where:

**scheme**

The scheme of the URL (either HTTP: or HTTPS:).

**hostname**

The TCP/IP hostname or numeric IP address of the TADDM server.

**port**

The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).

**guid**

The globally unique identifier (GUID) of a model object instance that exists in the TADDM database. If you are updating an object, this GUID must match the GUID specified in the JSON or XML object data.

HTTP methods

**GET**  Queries a model object.

**PUT**  Updates a model object. The new object data must be specified in the body of the HTTP request, in either JSON or XML format. (The server automatically detects the format of the input data.)

**DELETE**  Deletes a model object.

Parameters

**depth=value**

The depth of the query. The default value is 1. This parameter is not used when updating or deleting objects.

**feed={json|xml}**

The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

The feed parameter is not used when updating or deleting an object.
longClassName={true|false}
Specifies whether all model object class names in the output from a query are
specified using the fully qualified form (for example, com.collation.platform.model.topology.sys.ComputerSystem). Specify true or
false. This option is valid only for JSON output. The default value is false.

mssGuid=value
The GUID value of the management software system (MSS) associated with the
object. This parameter is optional.

Returns
If the request is successful, the server returns HTTP return code 200. For a query,
the server also returns the result data in either JSON or XML format (as specified
by the feed parameter or the HTTP Accept header). If the query returns no data,
the result set is an empty JSON array or XML document, depending on the feed
type.

Example
This example queries, updates, or deletes an existing object, depending on the
HTTP method used. (To update an object, the body of the request must contain the
updated object data.)
http://example.com:9430/rest/model/Object/1D646C44FDEB3857B40B98BD
F9C0F407?mssGuid=CF5EBF574E7F3822893F35FB5776628

Model object class metadata:
The model object class metadata resource represents the metadata describing the
attributes of a model object class.

Description
Use the model object class metadata resource to query data about the attributes of
a specified model object class, including the number, type, and name of each
attribute. This information is equivalent to that returned by the Java getMetaData() method.

The metadata can be returned in either JSON or XML format.

URL
scheme://hostname:port/rest/model/meta/model_object_class

where:
scheme
The scheme of the URL (either HTTP: or HTTPS:).
hostname
The TCP/IP hostname or numeric IP address of the TADDM server.
port
The TCP/IP port on the TADDM server for the type of connection you are
using (9430 for HTTP, or 9431 for HTTPS).
model_object_class
The name of a Common Data Model object class.
HTTP methods

GET  Queries object class metadata.

Parameters

feed={json|xml}

The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

Example

This example queries metadata information for the ComputerSystem model object:
http://example.com:9430/rest/model/meta/ComputerSystem?feed=json

This example shows JSON output from a metadata query:
```json
[{
"type": "java.lang.String",
"column": "BOOTORDER_X",
"length": 192,
"name": "bootOrder",
"arrayType": false,
"class": "ObjectAttribute",
"timestampType": false,
"displayText": "Boot Order"}, {
"type": "com.collation.platform.model.topology.sys.zOS.ZReportFile",
"table": "COMPUTERSYSTILES_935A6002X",
"column": "PK_ZREPORTFILES_X",
"length": 192,
"name": "ZReportfiles",
"arrayType": true,
"reverseRelationship": true,
"class": "ObjectAttribute",
"relationshipType": "com.collation.platform.model.topology.relation.AppliesTo",
"timestampType": false,
"displayText": "z/OS Report File"}]
```

This example shows partial XML output of a metadata query:
```
<ObjectAttribute array="22" xsi:type="coll:com.collation.platform.model.topology.meta.ObjectAttribute">
   <name>OSRunning</name>
   <type>com.collation.platform.model.topology.sys.OperatingSystem</type>
   <arrayType>false</arrayType>
   <timestampType>false</timestampType>
   <length>192</length>
   <relationshipType>com.collation.platform.model.topology.relation.RunsOn</relationshipType>
   <reverseRelationship>true</reverseRelationship>
   <displayString>OS Running</displayString>
   <column>PK_OSRUNNING_X</column>
   <displayName />
</ObjectAttribute>
```

Model object update service:

The model object update service resource creates, updates, or deletes model objects passed to the server in JSON or XML format.

Description

Use the model object update service to update or delete an existing model object, or to add a new model object. In each case, the target of the operation is the object specified in the body of the request, in JSON or XML format.
URL

scheme://hostname:port/rest/model/ModelObject

where:

scheme
  The scheme of the URL (either HTTP: or HTTPS:).

hostname
  The TCP/IP hostname or numeric IP address of the TADDM server.

port
  The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).

HTTP methods

POST
  Creates or deletes a model object, depending on the value of the delete parameter. The object to be created or deleted must be specified in the body of the HTTP request in JSON or XML format. (The server automatically detects the format of the input data.) Specify only one primary object; arrays of objects are not supported.

  If you use this method to create a new object, the specified object must not already exist in the TADDM database. (The POST method cannot be used to update an existing object.)

  If you use this method to delete an existing object, only the GUID is required in the input data. However, the entire object can also be specified. No error is returned if the specified object does not exist.

PUT
  Updates an existing object or creates a new object. The new object data must be specified in the body of the HTTP request in either JSON or XML format. Specify only one primary object; arrays of objects are not supported.

  If the specified object already exists, it is updated with the new data. If the object does not exist, it is created.

  If you are updating an existing object, you can improve performance by including only the GUID and the fields required for the update, instead of the entire object. For example, an update to the description of an OperatingSystem object might include the following data:

  ```json
  [{"description":"Validated on February 4","_class":"Linux","guid":"347EE64EFA93139A581757EC7F36D2D"}]
  ```

  Any object attributes not specified in the update data are left unchanged.

Parameters

delete={true|false}
  Indicates whether the specified model object should be deleted. Use the HTTP POST method and delete=true to delete an object.

feed={json|xml}
  The format to use for the returned data. Specify json or xml. This parameter is optional.
If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

mssGuid=value
The GUID value of the management software system (MSS) associated with the object. This parameter is optional.

Returns
If the request is successful, the server returns HTTP return code 200.

The following example deletes the model object specified by the input data:
http://example.com:9430/rest/model/ModelObject?delete=true

Discovery service:
The discovery service resource starts a discovery with or without a profile.

Description
Use this type of request to start a discovery using any currently defined profile, or without a profile.

URL
scheme://hostname:port/rest/discovery/start/run_name

where:
scheme
- The scheme of the URL (either HTTP: or HTTPS:).
hostname
- The TCP/IP hostname or numeric IP address of the TADDM server.
port
- The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).
run_name
- The name for the discovery run.

HTTP methods
POST Starts a discovery. A discovery must not already be in progress. You must submit the url request by using the HTTP POST operator.

Parameters
feed={json|xml}
The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

guids=values
One or more globally unique identifiers (GUIDs) of objects that have
previously been discovered. Use this parameter to run a rediscovery on existing objects, if you have enabled rediscovery.

**profile**=profile_name

The name of the profile to use. The specified profile must exist.

**scope**=values

One or more scopes, separated by commas. Each value can be any of the following:

- A defined scope name
- A specific IP address or host name (for example, 192.168.1.71 or server.example.com)
- A specific IP address to exclude, enclosed in parentheses (for example, (192.168.1.71))
- An IP address range (for example, 10.10.10.1-10.10.10.20)
- A subnet (for example, 10.10.20.0/255.255.255.0)

An IP address, address range, or subnet can be enclosed in parentheses to indicate that it should be excluded from the scope. For example, 192.168.1.1-192.168.1.72, (192.168.1.71) would include all IP addresses in the specified range except 192.168.1.71.

**Input example**

This example starts a discovery by using the Level 3 Discovery profile, with the scope including the hosts 192.168.100.101 and 102.168.100.102. You can use any tool or utility that can make an HTTP request and submit the request by using the POST operator.

http://example.com:9430/rest/discovery/start/TestRun2?profile=Level%203%20Discovery&scope=192.168.100.101,192.168.100.102

**Returns**

If the request is successful, the HTTP response code 200 is returned, along with the message Discovery start submitted. If a discovery is already in progress, the request fails and an error message is returned. You can then use the discovery status resource to monitor discovery progress.

**Discovery status:**

The discovery status resource represents the current discovery status on the TADDM server.

**Description**

Use this resource to check the status of the current discovery run. The information returned is equivalent to that returned by the Java getStatus() method.

**URL**

```
scheme://hostname:port/rest/discovery/status
```

where:

```
scheme
```

The scheme of the URL (either HTTP: or HTTPS:).
hostname
The TCP/IP hostname or numeric IP address of the TADDM server.

port
The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).

HTTP methods
GET Queries discovery status.

Parameters
feed={json|xml}
The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

Returns
The current discovery status is returned using the specified format. The following example shows discovery status in XML format:

```
<status>Idle</status>
```

Example
This example checks discovery status:

```
http://example.com:9430/rest/discovery/status?feed=xml
```

Discovery profile service:
The discovery profile service resource lists the defined discovery profiles.

Description
Use this resource to retrieve a list of all discovery profiles currently defined on the TADDM server.

URL

```
scheme://hostname:port/rest/discovery/profiles
```

where:

```
scheme
The scheme of the URL (either HTTP: or HTTPS:).
hostname
The TCP/IP hostname or numeric IP address of the TADDM server.
port
The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).
```
HTTP methods
GET    Lists discovery profiles.

Parameters
feed={json|xml}

The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

Returns
A list of defined discovery profiles is returned using the specified format. The following example shows output in JSON format:

```json
[{
"name": "profile1"}, {
"name": "profile2"
}]
```

Example
This example lists discovery profiles:

http://example.com:9430/rest/discovery/profiles?feed=json

Discovery profile:
The discovery profile resource represents a defined discovery profile.

Description
Use the discovery profile resource to retrieve detailed information about a defined discovery profile.

URL

```
scheme://hostname:port/rest/discovery/profile/profile_name
```

where:

- `scheme`
The scheme of the URL (either HTTP: or HTTPS:).
- `hostname`
The TCP/IP hostname or numeric IP address of the TADDM server.
- `port`
The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).
- `profile_name`
The name of a defined discovery profile.

HTTP methods
GET    Retrieves details of a discovery profile.

Parameters
feed={json|xml}
The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

Returns

The details of the discovery profile are returned using the specified format.

Example

This example retrieves information about the Level 3 Discovery profile:
http://example.com:9430/rest/discovery/profile/Level%203%20Discovery?feed=xml

Discovery scope service:

The discovery scope service resource lists the defined discovery scopes.

Description

Use this resource to retrieve a list of all discovery scopes currently defined on the TADDM server.

URL

scheme://hostname:port/rest/discovery/scopes

where:

scheme
The scheme of the URL (either HTTP: or HTTPS:).

hostname
The TCP/IP hostname or numeric IP address of the TADDM server.

port
The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).

HTTP methods

GET Lists discovery scopes.

Parameters

feed={json|xml}

The format to use for the returned data. Specify json or xml. This parameter is optional.

If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

Returns

A list of defined discovery scopes is returned using the specified format. The following example shows output in JSON format:
Discovery scope:
The discovery scope resource represents a defined discovery scope.

Description
Use the discovery scope resource to retrieve detailed information about a defined discovery scope.

URL

scheme://hostname:port/rest/discovery/scope/scope_name

where:
scheme
  The scheme of the URL (either HTTP: or HTTPS:).
hostname
  The TCP/IP hostname or numeric IP address of the TADDM server.
port
  The TCP/IP port on the TADDM server for the type of connection you are using (9430 for HTTP, or 9431 for HTTPS).
scope_name
  The name of a defined discovery scope.

HTTP methods
GET    Retrieves details of a discovery scope.

Parameters

feed={json|xml}
  The format to use for the returned data. Specify json or xml. This parameter is optional.
  If you do not specify the feed parameter, the server uses the format specified by the HTTP Accept header (application/json or application/xml). If this header is not specified, the results are returned in JSON format.

Returns
The details of the discovery scope are returned using the specified format.

Example
This example retrieves information about the scope1 scope:
http://example.com:9430/rest/discovery/scope/scope1?feed=xml
Command-line interface API

You can use api.bat or api.sh to issue various commands on the TADDM server through the command-line interface (CLI). For example, you can use the CLI to start a discovery run.

Command syntax and parameters

You can use api.sh or api.bat to access a portion of the TADDM API functionality. The command syntaxes present the rules for running api.sh and api.bat.

For UNIX:

api.sh -u|--user user -p|--password password [-H|--host host] [-P|--port port]
COMMAND COMMAND-PARAMETERS

For Windows:

api.bat -u|--user user -p|--password password [-H|--host host] [-P|--port port]
COMMAND COMMAND-PARAMETERS

Common parameters

-u|--user user
   The user that runs the API command.

-p|--password password
   The password that authenticates the user.

-H|--host host
   Optional: The TADDM server host name, by default, is localhost.

-P|--port port
   Optional: The TADDM server port, by default, is 9530.

-v|--version version
   Optional: The version name or number, by default, is 0.

COMMAND COMMAND-PARAMETERS
   The parameters are different for each of the commands.

Additional information

To see help about the command and command-parameters, enter the following command from the $COLLATION_HOME/sdk/bin directory:

On UNIX systems
   api.sh

On Windows systems
   api.bat

Changes command

The changes command retrieves the changes for an object.

Command syntax

api.sh | api.bat -u|--user user -p|--password password [-H|--host host] [-P|--port port] changes guid from-date [to-date]
Parameters

changes
Runs the changes command.

guid
Is the GUID of the object for which you want to determine changes.

from-date
Is the beginning date of the changes command. Use the mm/dd/yy hh:mm:ss AM|PM format.

to-date
Is the end date of the changes command. Use the mm/dd/yy hh:mm:ss AM|PM format.

Example

This command finds all changes to an object that occurred between two specific dates. Enter the command on one line:

```
api.sh -u user -p password -H host -P port changes 10A5794E86C53A0BBB10F26205CB3EA "06/06/05 12:00:00 AM" "06/08/05 12:00:00 AM"
```

Delete command

The delete command removes an object from the TADDM database.

Command syntax

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] delete guid
```

Parameters

delete
Runs the delete command.

guid
Is the GUID of the object to delete.

Example

This command deletes an object with the specified GUID. Enter the command on one line:

```
api.sh -u user -p password -H host -P port delete 10A5794E86C53A0BBB10F26205CB3EA
```

Discover command

The discover command starts or stops a discovery run.

Command syntax

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] discover start [--name run-name] [--profile profile-name] scope-element1 scope-element2 ... scope-elementn

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] discover abort | status scope-element1 scope-element2 ... scope-elementn
```

SDK Developer's Guide
Parameters

discover
Runs the discover command.

start scope-element1 scope-element2 ... scope-elementn
Starts a discovery with the specified scope elements. The scope element can be an existing Scope name, or:
- Specific IP address: 192.168.1.71
- Exclude of a specific IP address: 192.168.1.71(exclude), or (192.168.1.71), or 192.168.1.71(exc)
- Range or Range Exclude: 10.10.10.1-10.10.10.20, or (10.10.10.1-10.10.10.20)
- Network (Subnet) or Network Exclude: 10.10.20.0/255.255.255.0, or (10.10.20.0/255.255.255.0)

--name run-name
Is the name of the discovery run.

--profile profile-name
Uses the profile that is specified by profile name for the discovery.

abort|stop
Stops a running discovery on the specified host.

status
Returns the discovery status on the specified host, from among the following values:
- Running
- Idle
- Waiting
- Aborted

Examples
- This command discovers subnet 10.10.10.0/24 using a Level 1 discovery profile. Enter the command on one line:
  api.sh -u user -p password -H host discover start
  --profile "Level 1 Discovery" "10.10.10.0/255.255.255.0"
- This command discovers the scope named MyScope using a Level 2 discovery profile. Enter the command on one line:
  api.sh -u user -p password -H host
discover start --profile "Level 2 Discovery" "MyScope"
- This command discovers the scope named MyScope using a Level 3 discovery profile with a 1.2.3.4 host excluded and 2.3.4.5-2.3.4.7 range included. Enter the command on one line:
  api.sh -u user -p password -H host discover start
  --profile "Level 3 Discovery" "MyScope" "(1.2.3.4)" "2.3.4.5-2.3.4.7"

Export command
The export command exports data for top-level model objects in the TADDM database.

Command syntax

Parameters

export
 Runs the export command.

--mssguid mss-guid|--mssname mss-name
 Is the GUID or the name of the Management Software System. Only data that is associated with the specified MSS is exported.

--maxfilesize size
 Is the maximum size of the exported files, in bytes.

local-directory-to-write-data
 Is the name of the directory to which the data is exported.

Example

This command exports top-level model objects to the specified directory:

api.sh -u user -p password -H host export directory/

Find command

The find command finds a set of objects and returns an XML representation.

Command syntax


api.sh | api.bat -u|--user user -p|--password password [-H|--host host] [-P|--port port] find [--depth depth] [--indent num-spaces] --changetype type --from from-date [--end end-date] root


Parameters

find
 Runs the find command.

--depth depth
 Is the level of the result tree to construct.

Querying a large amount of data or specifying more than one level can cause out of memory messages. To avoid memory issues, limit the depth value or increase the maximum heap size of the JVM memory. You can increase the memory with the-<em>-Xmx</em> JVM option in api.bat or api.sh.

--indent num spaces
 Is the indentation to use for the resulting XML output.

--changetype type
Is the type of change, from among the following values:

0 Created
1 Updated
2 Deleted
3 Creates and updates
4 All changes

--from from-date
Is the beginning date of the change parameter. Use the **mm/dd/yy hh:mm:ss AM|PM** format.

--end end-date
Is the end date of the change parameter. Use the **mm/dd/yy hh:mm:ss AM|PM** format.

-o|--outfile local-file-to-write-to
Is the name of the file to redirect the output of the **find** command to.

-x|--maxfilesize size
Is the outfile can be wrapped into several smaller files by specifying the maximum file size in bytes. The output is split into several files under the maximum file size, when possible.

-s|--suppress list-of-classes-to-suppress
Is a list of classes to be omitted from the find results. The classes are model object name classes, such as ComputerSystem or OperatingSystem.

--guid object-guid
Is the GUID of the object for which the **find** command is being executed.

--mssguid mss-guid|--mssname mss-name
Is the GUID or the name of the Management Software System.

--count mql-query
Returns the number of objects that meet the MQL query.

**mql-query**
Is the query that is specified using the Model Query Language (MQL), for example, **SELECT attributes FROM object type [WHERE expression]**. You can use long or short names for the object types in this argument. For more information about class names and MQL queries, see the related concepts.

**root**
Is the model object to serve as the root for the resulting XML output. You can use long or short names for the object types in this argument. For more information about class names, see the related concept.

**Examples**
- This command finds computer systems and saves the results to the **cs_output.xml** file with a maximum file size of 1000 bytes:

`api.sh -u user -p password -H host -P port find -o cs_output.xml -x 1000 ComputerSystem`

- This command counts the number of ComputerSystem objects in the database:

`api.sh -u user -p password find --count "select * from ComputerSystem"`

- This command limits the find to a defined depth level. The results are saved to the **cs_output.xml** file:

`api.sh -u user -p password -H host -P port find --depth depth -o cs_output.xml ComputerSystem`
**Import command**
The `import` command imports data into the TADDM database.

**Command syntax**
```
```

**Parameters**
- `import` Runs the `import` command.
- `--timeout time` Is the timeout value, useful for large file imports. Specify the value in seconds.
- `--mssguid mss-guid|--mssname mss-name` Is the GUID or the name of the Management Software System with which the imported data is associated.
- `local-directory-to-read-data-from` Is the name of the directory from which the data is imported.

**Example**
This command imports data into TADDM. The command attempts to import all files in the specified directory. If the command encounters an invalid XML file, it returns an exception but the command continues importing.
```
api.sh -u user -p password -H host import directory/
```

**Naming command**
The `naming` command returns the GUIDs that are associated with a configuration item (CI). The command returns only the GUIDs of top-level CIs in the XML file.

**Command syntax**
```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] -f model-object-xml-file
```

**Parameters**
- `naming` Runs the `naming` command.
- `-f model-object-xml-file` The location and name of the XML file that contains the configuration item (model object).

**Example**
This command displays the GUIDs for CIs in the XML file:
```
api.sh -u user -p password -H host naming sample.xml
```

**Servers command**
The `servers` command shows information about the servers in a streaming server deployment.

**Command syntax**
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getservers

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getdiscoveryservers

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getdiscoveryserverstatus

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getstorageservers

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getstorageserverstatus

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getlocalserver

api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] servers getlocalserverstatus

Parameters

servers
Runs the servers command.

getservers
Lists all running storage and discovery servers.

getdiscoveryservers
Lists all running discovery servers.

getdiscoveryserverstatus
Shows the detailed status and performance information for all discovery servers.

getstorageservers
Lists all running storage servers.

getstorageserverstatus
Shows the detailed status and performance information for all storage servers.

getlocalserver
Shows information about the local server.

getlocalserverstatus
Shows the detailed status and performance information for the local server.

Examples

- This command lists all running storage servers and discovery servers:
  api.sh -u user -p password -H host -P port servers getservers

- This command lists all running discovery servers:
  api.sh -u user -p password -H host -P port servers getdiscoveryservers

- This command shows the detailed status and performance information for all storage servers:
  api.sh -u user -p password -H host -P port servers getstorageserverstatus
Sync command

The `sync` command starts a domain server synchronization.

**Command syntax**

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] sync start domain
```

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] sync status domain
```

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] sync logs domain
```

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] sync stop domain
```

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] sync delete domain
```

**Parameters**

`sync`

- Runs the `sync` command.

`start`

- Starts a domain server synchronization.

`status`

- Shows the status of a domain server synchronization.

`logs`

- Shows the log files for a domain server synchronization.

`stop`

- Stops a domain server synchronization.

`delete`

- Deletes a domain server synchronization.

`domain`

- Is the name of the domain that is added to ECMDB Enterprise Server, not the host name of the domain server.

**Example**

This command starts a domain server synchronization:

```
api.sh -u user -p password -H host sync domain
```

Topology command

The `topology` command clears or rebuilds the topology.

**Command syntax**

```
api.sh | api.bat -u | --user user -p | --password password [-H | --host host] [-P | --port port] topology groups
```
Parameters

**topology**
Runs the `topology` command.

**groups**
Shows the detailed status of topology groups.

Example

This command shows the detailed status of topology groups:
```
api.sh -u user -p password topology groups
```

**Version command**

The `version` command manages versions in the CMDB.

Command syntax

```
api.sh | api.bat -u --user user -p --password password [-H --host host] [-P --port port] version [-c --create version-name version-description]
```

```
api.sh | api.bat -u --user user -p --password password [-H --host host] [-P --port port] version [-e --createempty version-name version-description]
```

```
api.sh | api.bat -u --user user -p --password password [-H --host host] [-P --port port] version [-d --delete version-id-or-name]
```

```
api.sh | api.bat -u --user user -p --password password [-H --host host] [-P --port port] version getall
```

Parameters

**version**
Runs the `version` command.

```
-c --create version-name version-description
```

Creates a new version with the supplied name.

```
-e --createempty version-name version-description
```

Creates an empty new version with the supplied name.

```
-d --delete version-id-or-name
```

Deletes the specified version.

```
getall
```
Displays all existing versions.

Examples

- This command creates a version:
  ```
  api.sh -u user -p password -H host -P port version -create "version1.0"
  "This is the initial version"
  ```

- This command deletes a version:
  ```
  api.sh -u user -p password -H host -P port version -delete "version1.0"
  ```

**Developing custom server extensions**

You can use custom server extensions to discover targets for which TADDM has limited or no specific built-in support or to add functionality to TADDM.
The custom server extensions provide an Application Programming Interface (API) that you can use to create programs that set attributes defined in the Common Data Model (CDM) or extended attributes you have added to the CDM using the Data Management Portal.

These Jython-based extensions run inside the TADDM Discovery Engine which provides the custom server extensions with a framework to harness many of the sensor building blocks within TADDM.

Custom server extensions offer the following features:

- You can use the TADDM user interface and API to view the discovered attributes.
- Custom server extension messages are written to the Discovery Manager log and to the appropriate computer system sensor log (or CustomAppServerSensor logs if split sensor logs are enabled).
- No additional software is required to use the system.

**Overview**

You can use the Data Management Portal and custom server extensions API to set built-in or extended attributes.

Developing a custom server extension involves identifying the built-in and extended attributes you want to set and adding the extended attributes to the Common Data Model. After this is done, you need to write the application to set the attributes and check that the attributes are being collected as expected.

The following outlines the procedure for developing a custom server extension:

1. Identify the built-in or extended attributes you want to collect.
2. If you identified extended attributes, add the attributes to the Common Data Model using the Data Management Portal.
   
   For more information, see "Managing extended attributes."
3. Develop the application to set the attributes using the custom server extensions API.
   
   For more information about the custom server extensions API, see "Custom server extensions API" on page 96. To see a sample application, refer to "Sample custom server extension application" on page 124.
4. Run the custom server extension application.
5. Using the Data Management Portal, verify that the attributes are being set as expected.

**Managing extended attributes**

You need to define the extended attributes before you can collect the attributes using your custom server application.

**About this task**

You can use the Data Management Portal to add or delete extended attributes for a component type in the Common Data Model.

Table 30 on page 96 describes the settings that you can specify for extended attributes.
Table 30. Extended attributes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component type</td>
<td>The type of component.</td>
</tr>
<tr>
<td>Extended attribute name</td>
<td>The name of the extended attribute.</td>
</tr>
<tr>
<td>Extended attribute type</td>
<td>The type for the extended attribute.</td>
</tr>
<tr>
<td>Inherited attribute name</td>
<td>If this class is a subclass of another class in the Common Data Model, and if extended attributes have been defined for the parent class, these attributes are listed here.</td>
</tr>
<tr>
<td>Inherited attribute type</td>
<td>The type of the inherited attribute.</td>
</tr>
</tbody>
</table>

Procedure

To specify extended attributes, complete the following steps:

1. Launch the Domain Management Portal.
2. Choose Edit > Extended Attributes from the main menu.
3. Choose a component type from the Component type list. The Define Extended Attributes window displays the currently defined extended attributes for the selected component type.
4. To add an extended attribute, click New. Enter the attribute name and the attribute type in the corresponding fields and click OK. The system adds the attribute name to the list of extended attributes.
5. To delete an extended attribute, complete the following steps:
   a. Select the corresponding component type from the Component type list.
   b. Select the attribute you want to remove in the Define Extended Attributes window.
   c. Click Delete.
6. Click OK to save the changes and dismiss the window, or click Cancel to dismiss the window without saving your changes.

Custom server extensions API

The custom server extensions API provides a set of functions that you can use to create Python applications that retrieve information about running processes, run commands, capture files, normalize data, create new Common Data Model objects, and set attributes or extended attributes in the Common Data Model.

This section describes what you need to do before starting to use the custom server extensions API, and then provides an overview of the types of functions available in the API. The section also includes a description of each class of function available, along with sample code that shows the most common elements you need to include in your applications.

Prerequisites to using the custom server extensions API

You must understand key concepts before using the custom server extensions API.

Before you create applications using the custom server extensions API, you should understand the following concepts:

- Common Data Model (CDM)
If you are using the custom server extensions to create new CDM ModelObjects, you need to set at least one naming rule otherwise TADDM will not be able to generate a GUID for the object and the sensor in which the extension runs will fail (issuing a Storage Error).

- How to have the application set up the Jython environment for use with TADDM and the custom server extensions API.
  You can use the sensorstub.py file in the $COLLATION_HOME/lib/sensor-tools directory as the basis for your custom server extension application. This code acts as a stub which sets up the Jython environment correctly but performs no operations on the system.
- How to create and manage custom servers.
  For more information, see the TADDM User’s Guide.

**Function overview**
The custom server extensions API offers several classes of functions to help you write custom server extensions.

The custom server extensions API consists of a set of Python functions that you can use to run commands and manage processes, perform DNS lookups, and gain access to directories and files on remote targets. The API also provides functions to manipulate media access control (MAC) and IP addresses and use operating system handles to retrieve information.

The API further supplies a set of utility functions that you can use to perform useful tasks within your applications.

The following categories of functions are available in the custom server extensions API.

**Capability**
  Use capabilities like ExecuteCapability, MibQueryCapability, or OsInfoCapability.

**Command and process**
  Run commands on a target as well as manage and display process-related information.

**DNS and domains**
  Perform domain name lookups and validate fully qualified domain names.

**File access**
  List directory contents and capture files from remote targets.

**IP and MAC address**
  Manipulate and convert IP and MAC addresses.

**Operating system**
  Create and use operating system handles to retrieve information.

**Path**
  Convert Windows and Unix path separator characters.

**Utility**
  Initialize the custom server API and perform miscellaneous useful tasks.

**Version information**
  Determine the API version numbers.
Capability functions:

Capability functions enable you to use capabilities like ExecuteCapability, MibQueryCapability, or OsInfoCapability. Using capability functions makes it easier for you to perform a required operation on a specified target.

You can use the capabilities function to retrieve the factory responsible for creating capabilities for a specified target. Table 31 describes the functions you can use.

Table 31. Capability functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSimpleCapabilitiesFactory</td>
<td>Returns SimpleCapabilitiesFactory for a given IP address.</td>
</tr>
</tbody>
</table>

Command and process functions:

Command and process functions enable you to run commands on a target as well as manage and display process-related information.

You can use the command and process functions to run a command on a target, optionally specifying a timeout value that determines how long the command is permitted to run. You can also use the functions to add a runtime process to the process pool and return the connection map, port list, runtime process map, and server processes associated with process identifiers.

Table 32 describes the functions you can use.

Table 32. Command and process functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addProcessToPool</td>
<td>Add a runtime process to a process pool.</td>
</tr>
<tr>
<td>executeCommand</td>
<td>Run a command on the target.</td>
</tr>
<tr>
<td>executeCommandWithTimeout</td>
<td>Run a command on the target with a timeout that specifies how long the command is permitted to run.</td>
</tr>
<tr>
<td>getPidConnectionMap</td>
<td>Return a Python dictionary of Python lists containing the following information:</td>
</tr>
<tr>
<td></td>
<td>• keys: process IDs</td>
</tr>
<tr>
<td></td>
<td>• lists: TCP connections of the process IDs</td>
</tr>
<tr>
<td>getPidPortList</td>
<td>Return a Python dictionary of Python lists containing the following information:</td>
</tr>
<tr>
<td></td>
<td>• keys: process IDs</td>
</tr>
<tr>
<td></td>
<td>• lists: the ports the process is using either for listening or connecting</td>
</tr>
<tr>
<td>getPidToRuntimeProcessMap</td>
<td>Return a Python dictionary containing the following information:</td>
</tr>
<tr>
<td></td>
<td>• process IDs</td>
</tr>
<tr>
<td></td>
<td>• runtime process information</td>
</tr>
<tr>
<td>getProcessByPid</td>
<td>Return the CDM RuntimeProcess object associated with a given process ID.</td>
</tr>
</tbody>
</table>
Table 32. Command and process functions (continued)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getServerProcesses</td>
<td>Return a Python dictionary of Python lists containing the following information:</td>
</tr>
<tr>
<td></td>
<td>• keys: process IDs</td>
</tr>
<tr>
<td></td>
<td>• lists: bind addresses of the listen ports</td>
</tr>
</tbody>
</table>

Common Data Model functions:

Common Data Model functions enable you to manage the Common Data Model.

You can use the Common Data Model (CDM) functions to create and clone new CDM objects and set the value of extended attributes in CDM objects.

Table 33 describes functions you can use.

Table 33. Common Data Model functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cloneModelObject</td>
<td>Create a copy of a CDM ModelObject.</td>
</tr>
<tr>
<td>newModelObject</td>
<td>Create a CDM object.</td>
</tr>
<tr>
<td>setExtendedAttributes</td>
<td>Set the values of the extended attributes.</td>
</tr>
</tbody>
</table>

DNS and domain functions:

DNS and domain functions enable you to perform domain name lookups and validate fully qualified domain names.

You can use the DNS and domain functions to perform name lookups of the TADDM server and names extracted from a remote configuration. You can also use the functions to validate a fully qualified domain name (FQDN).

Table 34 describes functions you can use.

Table 34. DNS functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLocalDNSLookup</td>
<td>Perform a name lookup on the TADDM server.</td>
</tr>
<tr>
<td>getRemoteDNSLookup</td>
<td>Perform a lookup of a name extracted from a remote configuration which may not resolve on the TADDM server.</td>
</tr>
<tr>
<td>validateFqdn</td>
<td>Check an FQDN to ensure it conforms to the rules outlined in RFC 1035.</td>
</tr>
</tbody>
</table>

File access functions:

File access functions enable you to list directory contents and capture files from remote targets.

You can use the file access functions to list the contents of a directory and to capture the contents and metadata of files on remote targets. Table 35 on page 100 describes the functions you can use.
Table 35. File access functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getFile</td>
<td>Capture a file from a remote target and return the file contents and metadata.</td>
</tr>
<tr>
<td>getFileWithLengthLimit</td>
<td>Capture a file, up to the specified maximum length, from a remote target and return the file contents and metadata.</td>
</tr>
<tr>
<td>listDirectory</td>
<td>Return a Python list containing the contents of a directory on a remote target.</td>
</tr>
</tbody>
</table>

IP and MAC address functions:

IP and MAC address functions enable you to manipulate and convert IP and MAC addresses.

You can use the IP and MAC address functions to manipulate MAC addresses, validate IP addresses, and convert IP addresses between different representations. Table 36 describes the functions you can use.

Table 36. IP and MAC address functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binToDot</td>
<td>Convert a binary representation of an IP address to dot notation.</td>
</tr>
<tr>
<td>bitsMaskToDottedDecimalMask</td>
<td>Convert network bits mask notation to dotted decimal notation, for example 24 to 255.255.255.0.</td>
</tr>
<tr>
<td>calcNetworkAddress</td>
<td>Calculate the network address given an IP address and netmask.</td>
</tr>
<tr>
<td>canonicalMac</td>
<td>Remove separators or radix notation from a MAC address and return the hexadecimal MAC address as a string with alpha characters capitalized.</td>
</tr>
<tr>
<td>classlessNotation</td>
<td>Calculate the classless notation of an IP network.</td>
</tr>
<tr>
<td>dotToBin</td>
<td>Convert an IPv4 address from dot notation to binary form.</td>
</tr>
<tr>
<td>ipInSubnet</td>
<td>Determine if an IP address is a member of a given subnet and not the broadcast address.</td>
</tr>
<tr>
<td>networkToList</td>
<td>Return all IP addresses that are members of the CDM representation of the IpNetwork parameter.</td>
</tr>
<tr>
<td>validateIp</td>
<td>Validate an IP address in dot notation.</td>
</tr>
</tbody>
</table>

Operating system functions:

Operating system functions enable you to create and use operating system handles to retrieve information.

You can use the operating system functions to create operating system handles and retrieve information using these handles. Table 37 on page 101 describes the functions you can use.
Table 37. Operating system functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAppTarget</td>
<td>Return the following information:</td>
</tr>
<tr>
<td></td>
<td>• the operating system handle for the target</td>
</tr>
<tr>
<td></td>
<td>• the result object for the sensor</td>
</tr>
<tr>
<td></td>
<td>• the application server object for the target</td>
</tr>
<tr>
<td></td>
<td>• the process environment for the target</td>
</tr>
<tr>
<td></td>
<td>• the seed object that caused the discovery engine to spawn the target</td>
</tr>
<tr>
<td>getCsTarget</td>
<td>Return the following information:</td>
</tr>
<tr>
<td></td>
<td>• the operating system handle for the target</td>
</tr>
<tr>
<td></td>
<td>• the result object for the sensor</td>
</tr>
<tr>
<td></td>
<td>• the computer system object for the target</td>
</tr>
<tr>
<td></td>
<td>• the process environment for the target</td>
</tr>
<tr>
<td></td>
<td>• the seed object that caused the discovery engine to spawn the target</td>
</tr>
<tr>
<td>getComputerSystem</td>
<td>Return an object to which the OS handle is connected with the attributes populated.</td>
</tr>
<tr>
<td>getNewOsHandle</td>
<td>Attempt to create a new OS handle to the specified target. This can be used to communicate with a machine other than the one for which the custom sensor is originally launched.</td>
</tr>
<tr>
<td>getOperatingSystem</td>
<td>Return an object representing the operating system to which the OS handle is connected.</td>
</tr>
<tr>
<td>queryRegistry</td>
<td>Return a registry key as XML.</td>
</tr>
</tbody>
</table>

Path functions:

Path functions enable you to convert Windows and Unix path separator characters.

You can use the path functions to substitute Microsoft Windows and Unix path separator characters. Table 38 describes the functions you can use.

Table 38. Path functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unixSlashes</td>
<td>Substitute Windows path separator characters for Unix path separators</td>
</tr>
<tr>
<td>windowsSlashes</td>
<td>Substitute Unix path separator characters for Windows path separators</td>
</tr>
</tbody>
</table>

Utility functions:

Utility functions enable you to initialize the custom server extensions API and perform miscellaneous useful tasks.

You can use the utility functions to initialize the custom server extensions API and perform useful tasks such as creating an array in Python for use with certain Java and TADDM functions as well as splitting command lines into their components.

Table 39 on page 102 describes the functions you can use.
**Table 39. Utility functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getArray</td>
<td>Get an array in Python for use with certain Java and TADDM functions.</td>
</tr>
<tr>
<td>init</td>
<td>Initialize the custom server extension API.</td>
</tr>
<tr>
<td>splitArgs</td>
<td>Split a command line into its components and return them as a Python sequence.</td>
</tr>
</tbody>
</table>

**Version information functions:**

Version information functions enable you to determine the API version numbers.

You can use the version information functions to determine the major and minor version numbers for the custom server extensions API, as well as the TADDM version number. Table 40 describes the functions you can use.

**Table 40. Version information functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getApiMinorVersion</td>
<td>Return the minor version of the API.</td>
</tr>
<tr>
<td>getApiVersion</td>
<td>Return the major version of the API.</td>
</tr>
<tr>
<td>getTADDMVersion</td>
<td>Return the TADDM version number.</td>
</tr>
</tbody>
</table>

**Function reference**

This reference describes each function that is available in the custom server extensions API. The functions are listed in alphabetical order.

**addProcessToPool function:**

Add a runtime process to a process pool.

**Description**

The addProcessToPool function adds a Common Data Model (CDM) RuntimeProcess object to a ProcessPool. You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

**Function syntax**

`addProcessToPool (rp, pool, *os)`

**Parameters**

- `rp` The CDM RuntimeProcess object
- `pool` The CDM ProcessPool object
- `*os` (Optional) The OS handle object

**Returns**

The function returns the OS handle object connected to the new target.
Exceptions
OsException.

binToDot function:
Convert a binary representation of an IP address to dot notation.

Description
The binToDot function converts a binary representation of an IP address to dot notation.

Function syntax
binToDot (binIp)

Parameters
binIp
A Python long containing the binary representation of the IP address

Returns
The function returns the string representation of an IP network address in dot notation.

Exceptions
None.

bitsMaskToDottedDecimalMask function:
Convert network bits mask notation to dotted decimal notation.

Description
The bitsMaskToDottedDecimalMask function converts a network bits mask notation representation to a dotted decimal representation, for example 24 to 255.255.255.0. Note that you should omit the leading / (slash) in the bits mask. The valid bits counts are 8 and 16-32.

Function syntax
bitsMaskToDottedDecimalMask (bits)

Parameters
bits
The string representation of the number of network bits

Returns
The function returns the dotted decimal form of the address.
Exceptions

None.

calcNetworkAddress function:

Calculate the network address given an IP address and netmask.

Description

The calcNetworkAddress function calculates the network address using the specified IP address and netmask.

Function syntax

calcNetworkAddress (ip)

Parameters

ip  The string representation of the IP address

(mask)
    The string representation of the subnet mask

Returns

The function returns the string representation of the IP network address.

Exceptions

None.

canonicalMac function:

Remove separators or radix notation from a MAC address and return the hexadecimal MAC address.

Description

The canonicalMac function removes separators or radix notation from a MAC address and returns the hexadecimal MAC address as a string with alpha characters capitalized.

Function syntax

canonicalMac (mac)

Parameters

mac  The MAC address

Returns

The function returns a string representation of the canonical MAC address.
Exceptions
None.

classlessNotation function:
Calculate the classless notation of an IP network.

Description
The classlessNotation function calculates the classless notation of an IP network.

Function syntax
classlessNotation (ip, mask)

Parameters
ip  The string representation of the IP network
mask  The string representation of the subnet mask

Returns
The function returns the classless notation in string form.

Exceptions
None.

cloneModelObject function:
Create a copy of a Common Data Model ModelObject.

Description
The cloneModelObject function creates a copy of a Common Data Model (CDM) ModelObject, recursing indefinitely through any child ModelObject attributes.

Function syntax
cloneModelObject (mo)

Parameters
mo  The CDM ModelObject to clone

Returns
The function returns the clone of the CDM ModelObject.

Exceptions
None.
dotToBin function:
Convert an IPv4 address from dot notation to binary form.

Description
The dotToBin function converts an IPv4 address from dot notation to binary form.

Function syntax
dotToBin (ip)

Parameters
ip A string representation of an IP address

Returns
The function returns the IP address in binary form as a Python long type.

Exceptions
NumberFormatException if the IP address is not valid.

executeCommand function:
Run a command on the target.

Description
The executeCommand function runs a command on the target using the default command timeout of two minutes. Note that the function prepends the PATH setting for the target type as specified in the collation.properties file.

You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax
executeCommand (cmd, *os)

Parameters
 cmd The command string to run
*os (Optional) The OS handle object

Returns
The function returns a string containing the output of the command.

Exceptions
OsException.
executeCommandWithTimeout function:

Run a command on the target with a timeout that specifies how long the command is permitted to run.

Description

The executeCommandWithTimeout function runs a command on the target, with a timeout that specifies how long the command is permitted to run. Note that the function prepends the PATH setting for the target type as specified in the collation.properties file.

You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax

executeCommandWithTimeout (cmd, timeout, *os)

Parameters

cmd
    The command string to run

timeout
    The time the command is allowed to run (in milliseconds)

*os
    (Optional) The OS handle object

Returns

The function returns a string containing the output of the command.

Exceptions

OsException.

getApiMinorVersion function:

Return the minor version of the API.

Description

The getApiMinorVersion function returns the minor version of the custom server extension API.

Function syntax

getApiMinorVersion

Parameters

None

Return

The function returns the minor version number of the API.
Exceptions
None.

getApiVersion function:
Return the major version of the API.

Description
The getApiVersion function returns the major version of the custom server extension API.

Function syntax
getApiVersion

Parameters
None

Return
The function returns the major version number of the API.

Exceptions
None.

getAppTarget function:
Return a tuple containing information about the application target.

Description
The getAppTarget function returns the following information:
• The operating system handle for the target
• The result object for the sensor
• The application server object for the target
• The process environment for the target
• The seed object that caused the discovery engine to spawn the target

Function syntax
getAppTarget (target)

Parameters
target
   The target map
Returns

Returns a tuple containing the OS handle to the target, the result object, the Common Data Model (CDM) AppServer, the process environment if any, and the seed object.

Exceptions

None.

getArray function:

Get an array in Python for use with certain Java and TADDM functions.

Description

The getArray function returns an array in Python for use with certain Java and TADDM functions. The function uses the Jython jarray module to create the array.

Function syntax

getArray (seq, classname)

Parameters

seq

The Python list or sequence

classname

The fully qualified Java class name matching the type of objects specified in the seq parameter

Returns

The function returns a Java array suitable for passing to Java methods which require an array.

Exceptions

None.

getComputerSystem function:

Return an object to which the OS handle is connected with the attributes populated.

Description

The getComputerSystem function returns the Common Data Model (CDM) ComputerSystem object to which the OS handle is connected with the attributes populated.

Function syntax

getComputerSystem (*os)
Parameters

*os
  (Optional) The OS handle object

Returns

The functions returns the CDM ComputerSystem object.

Exceptions

OsException.

getCsTarget function:

Return a tuple containing information about the target.

Description

The getCsTarget function returns a tuple containing the following information:
  • The operating system handle for the target
  • The result object for the sensor
  • The computer system object for the target
  • The process environment for the target
  • The seed object that caused the discovery engine to spawn the target

Function syntax

getCsTarget (target)

Parameters

target
  The target map passed to the custom server extension.

Returns

The function returns a tuple containing the OS handle to the target, the result object, the CDM ComputerSystem, and the seed object.

Exceptions

None.

getFile function:

Capture a file from a remote target and return the file contents and metadata.

Description

The getFile function captures a file from the remote target and returns a Common Data Model (CDM) FileSystemContent object containing the file contents and metadata.

Function syntax
**getFile** *(path, *os)*

**Parameters**

*path*

The path of the file to capture

*os*

(Optional) The OS handle object

**Returns**

The function returns the CDM FileSystemContent object containing the file contents and metadata.

**Exceptions**

OsException.

**getFileWithLengthLimit function:**

Capture a file, up to the specified maximum length, from a remote target and return the file contents and metadata.

**Description**

The getFileWithLengthLimit function captures a file from the remote target and returns a Common Data Model (CDM) FileSystemContent object containing the file contents and metadata.

You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

**Function syntax**

**getFileWithLengthLimit** *(path, length, *os)*

**Parameters**

*path*

The path of the file to capture

*length*

The maximum length to capture (in bytes)

*os*

(Optional) The OS handle object

**Returns**

The function returns a CDM FileSystemContent object containing the file contents and metadata.

**Exceptions**

OsException.
getLocalDNSLookup function:

Perform a name lookup on the TADDM server.

Description

The getLocalDNSLookup function performs a name lookup on the TADDM server and returns a Common Data Model (CDM) DNSLookup object containing the result. The function also accepts an optional OS handle but, regardless, the lookup is always local.

Function syntax

getLocalDNSLookup (name, *os)

Parameters

name
    The name to be resolved

*os
    (Optional) The OS handle object

Returns

The function returns the CDM DNSLookup object.

Exceptions

OsException.

getNewOsHandle function:

Create a new OS handle to the specified target.

Description

The getNewOsHandle function attempts to create a new OS handle to the specified target. You can use this to communicate with a machine other than the one for which the custom server extension is originally launched. An exception is raised if an SSH or WMI session cannot be established using the access lists currently configured in the TADDM server.

Function syntax

getNewOsHandle (ip)

Parameters

ip    The IP address of the machine to which you want to connect

Returns

The function returns the OS handle object connected to the new target.

Exceptions

OsException.
getOperatingSystem function:

Return an object representing the operating system to which the OS handle is connected.

Description

The getOperatingSystem function returns the Common Data Model (CDM) OperatingSystem object representing the operating system to which the OS handle is connected.

Function syntax

getOperatingSystem (*os)

Parameters

*os

The OS handle object

Returns

The function returns the CDM OperatingSystem object.

Exceptions

OsException.

getPidConnectionMap function:

Return a Python dictionary containing the process IDs and TCP connections of the process IDs

Description

The getPidConnectionMap function returns a Python dictionary of Python lists containing the process IDs as the keys and the lists of TCP connections of the process IDs.

You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax

getPidConnectionMap ( )

Parameters

None.

Returns

The function returns a Python dictionary of Python lists containing the following information:

- process IDs
- TCP connections of the process IDs
Exceptions

None.

getPidPortList function:

Return a Python dictionary containing the process IDs and the ports the process is using either for listening or connecting.

Description

The getPidPortList function returns a Python dictionary of Python lists containing the process IDs as the keys and lists of ports the process is using either for listening or connecting.

You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax

getPidPortList (*os)

Parameters

*os

(Optional) The OS handle object.

Returns

The function returns a Python dictionary with the following information:

- process IDs
- Python lists of CDM BindAddress objects

Exceptions

OsException.

getPidToRuntimeProcessMap function:

Return a Python dictionary containing the process IDs and runtime process information.

Description

The getPidToRuntimeProcessMap function returns a Python dictionary with the keys containing the process IDs and the values representing the Common Data Model (CDM) RuntimeProcess objects. You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax

getPidToRuntimeProcessMap (*os)
Parameters
*os
    (Optional) The OS handle object.

Returns
The function returns a Python dictionary with the following information:
- process IDs
- CDM RuntimeProcesses

Exceptions
None.

getProcessByPid function:

Return the Common Data Model RuntimeProcess object associated with a given process ID.

Description
The getProcessByPid function returns the Common Data Model (CDM) RuntimeProcess object associated with the specified process ID. You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax
getProcessByPid (pid, *os)

Parameters
pid
    The process ID.

*os
    (Optional) The OS handle object

Returns
The function returns the CDM RuntimeProcess object or "None" if the process ID does not exist.

Exceptions
None.

getRemoteDNSLookup function:

Perform a lookup of a name extracted from a remote configuration which may not resolve on the TADDM server.

Description
The getRemoteDNSLookup function performs a name lookup on the system specified in the first parameter. You can use this function to resolve names extracted from
remote configurations that may not resolve on the TADDM server.

Function syntax

getRemoteDNSLookup (ip, name)

Parameters

ip  The IP address of the machine where the lookup is to occur
name The name to be resolved

Returns

The function returns the CDM DNSLookup object.

Exceptions

OsException.

getServerProcesses function:

Return a Python dictionary of Python lists containing the process IDs and bind addresses of the listen ports.

Description

The getServerProcesses function returns a Python dictionary of Python lists of Common Data Model (CDM) BindAddress objects. You can optionally pass an OS handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax

getServerProcesses (*os)

Parameters

*os (Optional) The OS handle object

Returns

Returns a Python dictionary containing the following information:
- process IDs
- CDM BindAddress objects for the listen ports of the process IDs

Exceptions

OsException.

getSimpleCapabilitiesFactory function:

Return SimpleCapabilitiesFactory for a given IP address.
Description

The getSimpleCapabilitiesFactory function returns a SimpleCapabilitiesFactory for a given IP address. SimpleCapabilitiesFactory can be used to retrieve the following capabilities:

**ExecuteCapability**
This capability allows a command to be run on a given target, regardless of the communication protocol used.

**MibQueryCapability**
This capability allows a MIB query to be run on a given target.

**OsInfoCapability**
This capability allows OS information to be retrieved on a given target.

For more information about these capabilities, see the Javadoc, located in the following locations:
- $COLLATION_HOME/sdk/doc/capabilities/capabilities-javadoc.zip on the TADDM server.
- sdk/sdk.zip/doc/capabilities/capabilities-javadoc.zip on the TADDM installation DVD.

Function syntax

```
getSimpleCapabilitiesFactory (ip)
```

**Parameters**

- `ip` An IpV4Address object representing the IP address of the target host.

**Return**

The function returns a SimpleCapabilitiesFactory.

**Exceptions**

IllegalArgumentException if the IP parameter is null or is not a valid IPv4 address.

getTADDMVersion function:

Return the TADDM version number.

Description

The getTADDMVersion function returns the version of TADDM used.

Function syntax

```
getTADDMVersion
```

**Parameters**

None

**Returns**

The function returns the version of TADDM used.
Exceptions

None.

**init function:**

Initialize the custom server extension API.

**Description**

The init function initializes the custom server extension API helper routines.

**Function syntax**

`init (target)`

**Parameters**

`targte`

The targets map

**Returns**

The function returns a tuple containing the following:

- The OS handle to the target
- The result object
- The CDM ComputerSystem or AppServer
- The seed object
- The logger for writing to the sensor log environment (if the target is an AppServer)

**Exceptions**

None.

**ipInSubnet function:**

Determine if an IP address is a member of a given subnet and not the broadcast address.

**Description**

The ipInSubnet function determines if an IP address is a member of a given subnet and not the broadcast address.

**Function syntax**

`ipInSubnet (ip, net, mask)`

**Parameters**

`ip` A string representation of an IP address

`net` A string representation of a network
The subnet mask of the network

**Returns**

The function returns the following:
- Non-zero if the IP address is a member of the subnet
- 0 if the IP address is not a member of the subnet

**Exceptions**

None.

**listDirectory function:**

Return a Python list containing the contents of a directory on a remote target.

**Description**

The listDirectory function returns a Python list of the contents of a directory on a remote target.

**Function syntax**

`listDirectory(path, *os)`

**Parameters**

- `path`
  - The path of the directory
- `*os`
  - (Optional) The OS handle object

**Returns**

The function returns the Python sequence of the contents of the directory.

**Exceptions**

OsException.

**networkToList function:**

Return all IP addresses that are members of the CDM representation of the IpNetwork parameter.

**Description**

The networkToList function returns all IP addresses that are members of the Common Data Model (CDM) representation of the IpNetwork parameter.

**Function syntax**

`networkToList(net)`
Parameters

net
    A CDM IpNetwork object

Returns

The function returns a Python list of string representations of IP addresses.

Exceptions

None.

newModelObject function:

Create a Common Data Model (CDM) object.

Description

The newModelObject function creates a new model object of the specified Common Data Model (CDM) class type.

Function syntax

newModelObject (classname)

Parameters

classname
    The fully qualified CDM class name of the CDM ModelObject to create

Returns

The function returns the new CDM ModelObject.

Exceptions

None.

queryRegistry function:

Return a registry key as XML.

Description

The queryRegistry function returns the requested registry key as XML. This function works only if the OS handle is connected to a Windows target; otherwise the function throws an exception.

You can optionally pass an operating system handle to the function. Otherwise, the OS handle passed in the targets map to the init function is used by default.

Function syntax

getOperatingSystem (key, *os)
Parameters

- **key**
  - The registry key to fetch

- **os**
  - (Optional) The OS handle object

Returns

The function returns the XML representation of the registry key

Exceptions

OsException and MethodNotImplementedException.

**setExtendedAttributes function:**

Set the values of the extended attributes.

Description

The setExtendedAttributes function accepts a Common Data Model (CDM) ModelObject and a Python dictionary of name-value pairs and sets the name-value pairs as extended attributes for the ModelObject.

Function syntax

```python
setExtendedAttributes (mo, exattrs)
```

Parameters

- **mo**
  - The Common Data Model ModelObject

- **exattrs**
  - The Python dictionary of name-value pairs where the name is the extended attribute name and the value is a string

Exceptions

IoException.

**splitArgs function:**

Split a command line into its components and return them as a Python sequence.

Description

The splitArgs function splits a command line into its components and returns them as a Python sequence.

Function syntax

```python
splitArgs (cmdline)
```
Parameters

CMDLINE
    A command line (the parameter must be quoted if it contains embedded
    spaces)

Returns

The function returns a Python sequence containing the command line tokens.

Exceptions

None.

unixSlashes function:

Converts Windows path separator characters to UNIX path separator characters.

Description

The unixSlashes function converts Windows path separator characters to UNIX
path separator characters.

Function syntax

unixSlashes(path)

Parameters

path
    A file system path which can contain Windows path separators

Returns

The function returns a file system path containing only UNIX path separators.

Exceptions

None.

validateFqdn function:

Check a full qualified domain name to ensure it conforms to the rules outlined in
RFC 1035.

Description

The validateFqdn function checks a full qualified domain name (FQDN) to ensure
that it conforms to the rules specified in RFC 1035. Note that the Discovery
Management Console does not display non-conforming FQDNs.

Function syntax

validateFqdn fqdn
Parameters

fqdn
   The fully qualified domain name

Returns

The function returns the following values:
• Non-zero if the FQDN is valid
• 0 if the FQDN is not valid

Exceptions

None.

validateIp function:

Validate an IP address in dot notation.

Description

The validateIp function validates that a IP address in dot notation is a valid IP address.

Function syntax

validateIp (ip)

Parameters

ip  A string representation of an IP address in dot notation

Returns

The function returns the following values:
• Non-zero if the IP address is valid
• 0 if the IP address is not valid

Exceptions

None.

windowsSlashes function:

Converts UNIX path separator characters to Windows path separator characters.

Description

The windowsSlashes function converts UNIX path separator characters to Windows path separator characters.

Function syntax

windowsSlashes (path)
Parameters

path
A file system path which can contain UNIX path separators

Returns

The function returns a file system path containing only Windows path separators.

Exceptions

None.

Sample custom server extension application
A typical custom server extension application includes several common segments of code.

The following sample custom server extension application shows the standard elements and code segments that you can include in your applications:

```python
import sys
import java

from java.lang import System

coll_home = System.getProperty("com.collation.home")
System.setProperty("jython.home",coll_home + "/external/jython-2.1")
System.setProperty("python.home",coll_home + "/external/jython-2.1")

jython_home = System.getProperty("jython.home")
sys.path.append(jython_home + "/Lib")
sys.path.append(coll_home + "/lib/sensor-tools")
sys.prefix = jython_home + "/Lib"

import traceback
import string
import re
import jarray
import sensorhelper

############################
# LogError Error logger
############################
def LogError(msg):
    log.error(msg)
    (ErrorType, ErrorValue, ErrorTB) = sys.exc_info()
    traceback.print_exc(ErrorTB)

############################
# main
############################
try:
    (os_handle, result, appserver,seed,log,env) = sensorhelper.init(targets)

    response = sensorhelper.executeCommand("ssh -V 2>&1")

    if response != None:
        match = re.search("OpenSSH_(\[^,\]+)",response)

        if match != None:
            appserver.setProductVersion(match.group(1))
            appserver.setProductName("OpenSSH")
            appserver.setVendorName("openssh.org")
        else:
```
log.info("This ssh server does not appear to be OpenSSH")
else:
    log.info("'ssh -V' returned no output")
except:
    LogError("unexpected exception getting ssh information")

**Explanation of the segments in the sample application**

This section describes the segments of the sample custom server extension application.

**Initializing the environment**

This section of code sets up the environment so that the Jython interpreter can find the standard Python modules and TADDM sensor tools Python module.

```python
from java.lang import System
coll_home = System.getProperty("com.collation.home")
System.setProperty("jython.home", coll_home + "/external/jython-2.1")
System.setProperty("python.home", coll_home + "/external/jython-2.1")
```

```python
jython_home = System.getProperty("jython.home")
sys.path.append(jython_home + "/Lib")
sys.path.append(coll_home + "/lib/sensor-tools")
sys.prefix = jython_home + "/Lib"
```

**Importing sensorhelper**

This section of code imports the TADDM sensor tools Python module. This code enables the application to call the custom server extension functions, for example, `sensortools.executeCommand("echo hello world")`.

```python
import sensorhelper
```

**Logging errors**

This section of code logs exception stack traces using the Python traceback module. For regular logging, you can use the log object returned by the `sensortools.init()` call.

```python
def LogError(msg):
    log.error(msg)
    (TypeError, ErrorValue, ErrorTB) = sys.exc_info()
    traceback.print_exc(ErrorTB)
```

**Initializing the sensorhelper**

This section of code initializes the sensor tools Python module with information about the target of the discovery that was passed to the custom server extension application by the TADDM discovery engine.

```python
(os_handle, result, appserver, seed, log, env) = sensorhelper.init(targets)
```

**Running the command**

This section of code calls the sensor tools `executeCommand()` function to run "ssh –V" on the discovery target. The resulting output of the command is stored in the response variable.

```python
response = sensorhelper.executeCommand("ssh -V 2>&1")
```

**Searching the response**

This section of code first checks the response variable to ensure that it is valid (for example, that the value is not None). The code then uses the Python regular expression module to parse the output from the `ssh -V` command stored in the response variable.
if response != None:
    match = re.search("OpenSSH_(\[^,\]+)",response)

Setting the attributes
This section of code first checks whether the response variable was successfully parsed by the Python regular expression module (the value of
the match variable is not equal to None). If the response variable was successfully parsed, the ssh version is stored in the productVersion
attribute of the Common Data Model (CDM) AppServer object. Additionally, the productName and vendorName attributes are set.
    if match != None:
        appserver.setProductVersion(match.group(1))
        appserver.setProductName("OpenSSH")
        appserver.setVendorName("openssh.org")

Setting the object
This section of code stores the CDM AppServer object in the Result object. After the sensor completes, all CDM ModelObjects contained by the sensor Result object are sent to the TADDM storage engine to be persisted in the database.
        result.setAppServer(appserver)

Best practices for developing custom server extension applications
You can optimize your custom server extension application by following a set of simple guidelines.

Use the following guidelines when developing custom server extension applications:
• Log the operations performed by the application.
    You can use the log object returned by the sensortools.init() function to perform logging operations.
• Use the Python traceback module to log exception stack traces.
    If you use the sensorstub.py file in the dist/lib/sensor-tools directory as the basis of your custom server extension application, the LogError function defined in the file performs this task.
• Increasing the number of model objects in the result object, increases the time required to store the objects.

TADDM database schema and views
The TADDM database contains all of the configuration items (CIs) managed by a TADDM domain.

The database can be populated with CIs in several different ways:
• TADDM discoveries
• Bulk loading of Discovery Library Adapter (DLA) book files
• Manual adding of CIs using the graphical user interface
• Programmatic adding of CIs through the TADDM application programming interface (API)

CIs in the TADDM database are organized according to their classification within the IBM Tivoli Common Data Model (CDM). The object types, attributes, relationships, and naming rules for the CDM are documented in the
CDMWebsite.zip file, located in the $COLLATION_HOME/sdk/doc/model directory. To browse this documentation, extract the .zip file to a new directory, and then open the misc/CDM.htm file in a Web browser.

For more information about the Tivoli Common Data Model, see "Introducing the Common Data Model" on page 1.

When developing custom reports for TADDM, you can use SQL queries to retrieve data stored in the TADDM database. However, rather than querying data directly from the TADDM database tables, reports should use the TADDM database views. TADDM provides many predefined database views to simplify the task of writing SQL queries to extract data from the database, and you can also design your own custom views.

There are three categories of TADDM database views:

- Building block views
- Details panel views
- Custom views

**Building block views**

You can use the building block views to write queries based on the Tivoli Common Data Model (CDM) point of view. These views are useful if you are familiar with the CDM; they do not require any knowledge of where configuration items are stored in the TADDM database base tables.

To see detailed documentation for the building block views, go to the $COLLATION_HOME/etc/views directory and open one of the following files:

- For DB2 databases: create_building_block_views_db2.sql
- For Oracle databases: create_building_block_views_oracle.sql

The comments in these files describe the Common Data Model object types and the corresponding database views, providing mappings between the Common Data Model and the database schema:

- from CDM object type to building block database view name
- from attribute to database view column name
- from relationship to JOIN syntax

The name of each building block view is in the following form:

```
BB_%_V
```

In each view name, the % is the name of the object type. Each object type is mapped to a building block view, yielding more than 1,000 available building block database views representing object types.

For example, the CDM defines the sys.windows.WindowsComputerSystem object type, which is stored in the COMPSYS base table in the TADDM database. This table also includes many other object types that extend the sys.ComputerSystem object type (for example, sys.LinuxUnitaryComputerSystem).

To query sys.windows.WindowsComputerSystem configuration items from the database using a building block view, you would query the BB_WINDOWSCOMPUTERSYSTEM20_V database view.
In addition to the views representing object types, more than 800 special views support "many-to-many" relationships between object types. Each of these mapping-table building block views has a name in the following form: BB_%J

For more information about these special views, see "JOIN definitions" on page [129]

**View definitions**

For each building block view, the comments include a section describing the view and the corresponding CDM objects. For example, the view definition section of the BB_WINDOWSCOMPUTERSYSTEM20_V building block view is as follows:

```
-- ######## model.topology.sys.windows.WindowsComputerSystem ########
--
-- View..... BB_WINDOWSCOMPUTERSYSTEM20_V
-- Class......................... model.topology.sys.windows.WindowsComputerSystem
-- Super classes................. model.topology.sys.ComputerSystem
-- model.topology.core.ManagedElement
-- model.ModelObject
-- model.topology.process.itil.ConfigurationItem
```

This example shows that the BB_WINDOWSCOMPUTERSYSTEM20_V view corresponds to the model.topology.sys.WindowsComputerSystem object type, and also lists several superclasses from which this type also inherits attributes and relationship definitions.

**Column definitions for attributes**

For each column corresponding to a CDM attribute, the comments include a section describing the column and the corresponding attribute. For example, the column definition section for the CPUSPEED_C column of the BB_WINDOWSCOMPUTERSYSTEM20_V view is as follows:

```
-- Column.... CPUSPEED_C
-- Attribute.................... CPUSpeed
-- Java Type.................... long, primitive
-- Declared By.................. model.topology.sys.ComputerSystem
```

This example shows that the CPUSPEED_C column corresponds to the CPUSpeed attribute defined by the model.topology.sys.ComputerSystem object type. (This attribute is inherited by the model.topology.sys.WindowsComputerSystem type.) It also lists the Java type used to represent the value of the attribute.

**Note:** The comments do not list the database type used to store the attribute. You can determine the database type by using a SQL describe command:

db2 describe table BB_WindowsComputerSystem20_V

**Column definitions for [0..1] relationships**

For each column representing a "zero or one" CDM relationship to another configuration item, the comments include a section describing the column used for performing the SQL JOIN operation to the configuration item on the other side of the relationship. For example, the column definition section for the PK_OSRUNNING_C column of the BB_WINDOWSCOMPUTERSYSTEM20_V view is as follows:
This example shows that the PK__OSRUNNING_C column is used to perform the JOIN operation to the OperatingSystem table, which represents the OSRunning relationship defined by the model.topology.sys.ComputerSystem object type. It also shows that the Java type of the attribute value is, in this case, another CDM type (model.topology.sys.OperatingSystem).

**Note:** All columns that represent relationships to other configuration items (and are therefore not primitive types) have names that start with PK__. From a relational database point of view, the value of such a column is the GUID of the configuration item on the other side of the relationship.

**JOIN definitions**

For both "zero or one" and "many-to-many" CDM relationship, the comments provide an example SQL query showing how to accomplish the SQL JOIN operation. For example, the JOIN definition section for the OSRunning relationship of the BB_WINDOWSCOMPUTERSYSTEM20_V view is as follows:

```
-- Column.... PK__OSRUNNING_C
-- Attribute..................... OSRunning
-- Java Type..................... model.topology.sys.OperatingSystem, notContained
-- Declared By................... model.topology.sys.ComputerSystem
-- Join......
-- Attribute..................... OSRunning
-- Java Type..................... model.topology.sys.OperatingSystem, notContained
-- Declared By................... model.topology.sys.ComputerSystem
-- Test Join..................... SELECT COUNT(1) FROM
-- BB_WINDOWSCOMPUTERSYSTEM20_V T1,
-- BB_OPERATINGSYSTEM62_V T2
-- WHERE T1.PK__OSRUNNING_C = T2.PK_C
```

Some relationships between configuration items are many-to-many relationships; for example, a sys.windows.WindowsComputerSystem configuration item might have a relationship to multiple sys.FileSystem configuration items, represented by a contains relationship using the fileSystems attribute of sys.windows.WindowsComputerSystem:

```
-- Join......
-- Attribute..................... fileSystems
-- Java Type..................... Array of model.topology.sys.FileSystem, array
-- Declared By................... model.topology.sys.ComputerSystem
-- Test Join..................... SELECT COUNT(1) FROM
-- BB_WINDOWSCOMPUTERSYSTEM20_V T1,
-- BB_COMPUTERSYSTEMTMS_88841D4BJ T2,
-- BB_FILESYSTEM71_V T3
-- WHERE T1.PK_C = T2.PK__JDOID_C
-- AND T3.PK_C = T2.PK__FILESYSTEMS_C
```

Many-to-many relationships are stored in the intermediary mapping table (in this example, the BB_COMPUTERSYSTEMTMS_88841D4BJ mapping table).

**Details pane views**

You can use the detail_panel_views.txt file to write queries and retrieve additional information. These views are most useful if you are familiar with the Details pane in the Data Management Portal.

To see a listing of available Details pane views, go to the $COLLATION_HOME/etc/views directory and open the detail_panel_views.txt file. This file lists all of the
Detail panes available in the Data Management Portal, along with the corresponding database views. The name of each Details pane view is in the following form:

\[ \text{DP}_{\%_{}}V \]

In each view name, the \% is the name of the Details pane in the Data Management Portal. There are more than 600 Details pane views.

Additional information about these views is available as comments in the following files in the same directory:

- For DB2 databases: create_detail_panel_views_db2.sql
- For Oracle databases: create_detail_panel_views_oracle.sql

**View definitions**

The Details pane views are organized according to the Common Data Model object types of the configuration items shown in the Details pane. To find the views for a particular configuration item, in the Details pane, click the **General** tab and find the value in the **Object Type** field. You can then find the object type in the detail_panel_views.txt file, which identifies the corresponding views.

For example, the detail_panel_views.txt file includes the following entry for the DB2 Instance object type:
This example shows that the General tab for the DB2 instance object type corresponds to the DP_DB2_INSTANCE_GENERAL_V database view. It also lists the additional database views available for the other tabs (some of which are represented by multiple views).

You can query these views to retrieve information about the configuration items that TADDM has discovered. For example, you can retrieve data from the Databases tab of DB2 Instance by querying the DP_DB2_INSTANCE_DATABASES_V database view. Selecting every element from this view returns the contents of the Database tab for all DB2 instances TADDM has discovered. You can limit this query to a single DB2 instance by joining the DP_DB2_INSTANCE_DATABASES_V view to the DP_DB2_INSTANCE_LICENSE_INFO_V view and filtering by the instance name (see “Example query” on page 132).

To see more information about how these views are defined, look at the comments in the create_detail_panel_views_db2.sql or create_detail_panel_views_oracle.sql scripts.
file. The following comment block provides information about the nodes, or CDM object types, referred to by the DP_DB2_INSTANCE_DATABASES_V view:

```sql
-- ######## Db2Instance.Databases ########
--
-- View...... DP_DB2_INSTANCE_DATABASES_V
--
-- Node...... 1
-- Node Path..................... Db2Instance
-- Node Class Name............... model.topology.app.db2.Db2Instance
-- Node Type..................... Root
-- Node...... 2
-- Node Path..................... Db2Instance._arraydatabases
-- Node Class Name............... model.topology.app.db2.Db2Database
-- Node Field Name............... databases
-- Node Type..................... Array Many-to-Many
-- Node...... 3
-- Node Path..................... Db2Instance._arraydatabases.databases
-- Node Class Name............... model.topology.app.db2.Db2Database
-- Node Field Name............... databases
-- Node Type..................... Array
```

**Nodes and columns**

To see how the view columns are defined, you can run a SQL describe query. For example, the following query shows the columns of the DP_DB2_INSTANCE_DATABASES_V view:

```sql
db2 "describe table DP_DB2_INSTANCE_DATABASES_V"
```

<table>
<thead>
<tr>
<th>Column name</th>
<th>Type schema</th>
<th>Type name</th>
<th>Length</th>
<th>Scale</th>
<th>Nulls</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME_C3</td>
<td>SYSIBM</td>
<td>VARCHAR</td>
<td>192</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>ALIAS_C3</td>
<td>SYSIBM</td>
<td>VARCHAR</td>
<td>192</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>PK_C3</td>
<td>SYSIBM</td>
<td>VARCHAR</td>
<td>192</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>PK_C1</td>
<td>SYSIBM</td>
<td>VARCHAR</td>
<td>192</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

Each column name ends with a numeral identifying the node the column represents. This example shows that the NAME_C3, ALIAS_C3, and PK_C3 columns all refer to the Db2Database object type, and the PK_C1 column refers to the Db2Instance object type.

**Note:** In this example, no column refers to node 2, because the referenced table is the many-to-many mapping table connecting DB2Instances with DB2Databases. The Details panel abstracts the complexities of joining these two tables, it is not necessary to determine where the data is stored in the base tables.

**Example query**

Using all of this information, you can see a list of DB2 databases defined for a particular instance by using this query:

```sql
select
    db.name_c3
from
    DP_DB2_INSTANCE_GENERAL_V inst
join DP_DB2_INSTANCE_DATABASES_V db ON (inst.PK_C1 = db.PK_C1)
where
    inst.db2_instance_c1 = 'bg-linux.tivlab.austin.ibm.com:db2inst1'
```
This query joins the views corresponding to the General and Databases tabs, by using the PK_C1 column (which represents the primary key of the DB2 instance) and selecting the database names. The results are then filtered to include only the instance labeled bg-linux.tivlab.austin.ibm.com:db2inst1.

**Custom views**

Custom views are provided to extract data from the TADDM database and to aid in the creation of reports. TADDM provides two custom views which are defined in the custom-views.xml file. You can also define views called user-defined views if the existing building block views and Details panel views do not provide what you need.

A custom view is defined by using XML, which is then processed by TADDM scripts to produce the required `CREATE VIEW` SQL statements. A custom view is built from existing building-block views, but the TADDM scripts determine how to join these views together.

TADDM includes a custom view called CM_COMPUTER_SYSTEMS_V, which is described in the following examples. This view provides basic information that can be used in a report about computer systems that TADDM has discovered:

- Fully qualified host name
- Manufacturer, model, serial number, and type of the chassis
- Type, number, and speed of the CPUs
- RAM size
- Operating system
- IP addresses of all adapters
- Total storage capacity and free space for all file systems

These attributes come from many different Common Data Model object types:

- ComputerSystem
- OperatingSystem
- IPInterface
- IPAddress
- FileSystem

In addition, the data includes two many-to-many relationships:

- ComputerSystem to IPInterface
- ComputerSystem to FileSystem

To manually write a query for this information, you must first identify all of the building-block views representing the CDM types and relationships. Then you must determine how to join them together. By defining a user view, you can use the TADDM scripts to automatically define the required joins and generate the SQL to create the views you require.

**User-defined views XML**

User-defined views like custom views are built from an xml definition. To create a user-defined view carry out the following steps:

1. Copy the custom-views.xml file from the $COLLATION_HOME/etc/views directory to the $COLLATION_HOME/bin directory.
2. Rename the file custom-views.xml as user-views.xml.
3. Change the view name CM_COMPUTER_SYSTEMS_V and CM_APP_SERVERS_PER_HOST_V in the user-views.xml file. These views are reserved by TADDM and must not be overwritten. Modify the rest of the file as required.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Contained elements</th>
</tr>
</thead>
</table>
| view      | className
   The base model object class name of the view.  
   viewName
   The name of the view. The name must be a string starting with CM_ and ending with _V, with a maximum length of 30 characters. Avoid names that are already in use.
   includePrimaryKeys
   Whether primary keys are included as columns. Must be true or false. Specify true if the view is to be joined with other views. | field |
| field     | None                                            | nested plain      |
| nested    | className
   The model object class name of the nested.  
   fieldName
   The field name of the nested | nested plain |

Application Dependency Discovery Manager: SDK Developer's Guide
The XML describes two types of fields, both of which are contained within the field element:

- A plain field represents an attribute of a model object. For example, the following plain field specifies that the FQDN (fully qualified domain name) attribute of ComputerSystem is displayed in the view as the FQDN column:

  ```xml
  <plain fieldName="fqdn" nameInView="FQDN"/>
  ```

- A nested field represents a relationship between model objects. For example, the following nested field describes the relationship from ComputerSystem to OperatingSystem through the OSRunning attribute of the ComputerSystem object type:

  ```xml
  <field>
    <nested className="com.collation.platform.model.topology.sys.OperatingSystem" fieldName="OSRunning">
      <plain fieldName="OSName" nameInView="OS_NAME"/>
    </nested>
  </field>
  ```

Within the nested field, a plain field specifies that the operating system name is displayed in the view as the OS_NAME column.

The full XML definition of the CM_COMPUTER_SYSTEMS_V view is as follows:

```xml
<views>
<!-- ComputerSystems with OS, Filesystems -->
<view className="com.collation.platform.model.topology.sys.ComputerSystem">
```
Adding user views to the database

After you have defined your user views in the user-views.xml file, follow these steps to create the views in the database:

1. At a command prompt, go to the $COLLATION_HOME/bin directory.
2. Run one of the following commands to create the required SQL scripts:
   - UNIX and Linux systems: user_views.sh scripts
   - Windows systems: user_views scripts
   This command creates the following files:
     - create_custom_views_db2.sql
     - create_custom_views_oracle.sql
     - create_custom_views_db2zos.sql
     - drop_custom_views_db2.sql
     - drop_custom_views_oracle.sql
     - drop_custom_views_db2zos.sql
3. Run one of the following commands to create the views in your database:
   - UNIX and Linux systems: user_views.sh recreate
   - Windows systems: user_views recreate
   This command runs the appropriate SQL script for your database type.

After you run these commands, your user views are available for querying in the database. Any SQL queries you implement must provide the necessary filtering of data returned from these views (for example, by using the SQL WHERE clause).

Modifying user views

To modify user views that exist in the database:
1. Edit the user-views.xml file to make the necessary changes.
2. Repeat the process of creating the views by using the user_views command.
   This command automatically generates and runs the correct SQL scripts to drop and then re-create any changed views.

   Note: If you rename a user view, you must manually drop the view with the original name before running the commands to create the view with the new name.

Deleting user views

To delete a user view from the database, run the appropriate SQL DROP command for your database. The DROP commands for the user views are generated by the user_views command in the following files:
- drop_custom_views_db2.sql
- drop_custom_views_oracle.sql
- drop_custom_views_db2zos.sql

Extended attributes views

The extended attributes view tool generates database views that reference the data for extended attributes.
For each model object that has extended attributes, the tool creates an SQL script that when run, creates an extended attribute view, `EA_model_V`, that has columns corresponding to the extended attributes. Each extended attribute can be joined to the corresponding building block view, `BB_model_V`, using the PK_C column.

All extended attributes on the same model object type are in the same database view.

The scripts depend on the current definitions of extended attributes. The tool does not check the attributes that were created and removed, even though such attributes with values are still assigned to some objects. If you make a change to an extended attribute, you must drop the existing view before using the extended attribute view tool to create an updated SQL script and an updated attribute view.

**The extattr_views.sh command syntax**

To use the extattr_view tool, run the `extattr_views.sh` command, with an appropriate command-line parameter. The `extattr_views.sh` command is in the `$COLLATION_HOME/bin` directory.

**Command syntax**

`extattr_views.sh parameter`

**Parameters**

`scripts`

Creates the following SQL scripts:

- `create_extattr_views_db_type.sql`
- `drop_extattr_views_db_type.sql`

where `db_type` is one of the following database types:

- `db2`
- `oracle`
- `db2zos`

`create`

Creates the views.

`remove`

Drops the views. The corresponding SQL scripts are not removed.

**Running the extended attributes view tool**

You can use the extended attributes view tool to generate a database view that corresponds to an existing extended attribute.

**Procedure**

To create a corresponding view for an extended attribute, complete the following steps:

1. Create an extended attribute on the model object.
2. Use the extended attributes view tool to create the required SQL scripts:
   
   `extattr_views.sh scripts`
3. Use the extended attributes view tool to create the view:
   
   `extattr_views.sh create`
4. Optional: Query the data using an SQL command.
Example

For example, if you create an extended attribute called ‘SUPPORT_AREA’ on the ComputerSystem model type, running the extended attributes view tool creates a view called EA_COMPUTERSYSTEM40_V. The new view has the following columns:

- PK_C, which is the primary key
- SUPPORT_AREA_C, which is the extended attribute

You can use the following SQL command to query the data:

```sql
SELECT T1.FQDN_C, T2.SUPPORT_AREA_C
FROM BB_COMPUTERSYSTEM40_V T1,
     EA_COMPUTERSYSTEM40_V T2
WHERE T1.PK_C = T2.PK_C
```

TADDM Data Dictionary

The TADDM Data Dictionary is a collection of automatically-generated HTML pages that provide a mapping between information in the Common Data Model (CDM) and information in the TADDM database.

Accessing the Data Dictionary

The Data Dictionary is available in the following locations on a storage server (in a streaming server deployment), a synchronization server (in a synchronization server deployment), or a domain server (in a domain server deployment):

- At http://taddmserverhost:9430/cdm/datadictionary/
- In the taddm-data-dictionary.zip file that is in the $COLLATION_HOME/sdk/datadictionary and $COLLATION_HOME/deploy-tomcat/cdm/datadictionary directories.

To use the taddm-data-dictionary.zip file, complete the following steps:

1. Extract the contents of the taddm-data-dictionary.zip file to a location of your choice.
2. Extract the subdirectories of the WebsiteFiles directory in the $COLLATION_HOME/sdk/doc/model/CDMWebsite.zip file to the data-dictionary/cdm directory of the extracted Data Dictionary structure.

Indexes

The Data Dictionary includes the following indexes:

Building Block Views Index

The index is available at:


From the index, click the name of a building block view. The following information is displayed:

- The TADDM database table from which the respective building block view is populated. Each table name links to a definition of the database table.
• Columns in the respective building block view. Each column name links to the CDM definition of the attribute that is represented by the column.

**Model Objects Index**

The index is available at:

From the index, click the name of a CDM class. The following information is displayed:
• TADDM database tables that contain the respective CDM class. Each table name links to a definition of the database table.
• Building block views that contain the respective CDM class. Each building block view name links to a definition of the building block view.

**Model Object Tables Index**

The index is available at:

From the index, click the name of a TADDM database table. The following information is displayed:
• The CDM class that declares the respective database table. The CDM class name links to a definition of the class.
• CDM classes that the respective database table contains. Each CDM class name links to a definition of the class.
• Columns in the respective database table. Each column name links to the CDM definition of the attribute that is represented by the column.

**Data Discovered By Sensors Index**

The index is available at:

From the index, click the name of a sensor. The following information is displayed:
• General information about the sensor.
• The CDM class of model objects that are discovered by the sensor. Each CDM class name links to a definition of the class.
• Attributes of the CDM classes and information about their availability in the context of the sensor.

**Potential Data To Discover Index**

The index is available at:

From the index, click the name of a category. The names of data types that belong to the respective category are displayed.

Click the name of a data type. The following information is displayed:
• General information about the sensor that discovers the respective data type
• The CDM class of model objects that are discovered by the sensor. Each CDM class name links to a definition of the class.
• Attributes of the CDM classes
Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX  78758  U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to
IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

### Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at [http://www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

[Java](http://www.java.com) and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.