Tivoli Application Dependency Discovery Manager
Version 7.3

SDK Developer's Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 163.

Edition notice

This edition applies to version 7, release 3 of IBM® Tivoli® Application Dependency Discovery Manager (product number 5724-N55) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Tables

About this information

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

Conventions used in this information center

In the IBM Tivoli Application Dependency Discovery Manager (TADDM) documentation certain conventions are used. They are used to refer to the operating system-dependent variables and paths, the `COLLATION_HOME` directory, and the location of the `collation.properties` file, which is referenced throughout the TADDM documentation, including in the messages.

Operating system-dependent variables and paths

In this information center, the UNIX conventions are used 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

TADDM root directory is also referred to as the `COLLATION_HOME` directory.

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

Refer to the following list of terms and definitions to learn about important concepts in the IBM Tivoli Application Dependency Discovery Manager (TADDM).

access collection
A collection that is used to control the access to configuration items and permissions to modify configuration items. You can create access collections only when data-level security is enabled.

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
A collection of components that provides a business functionality that you can use internally, externally, or with other business applications.

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.

Level 1 discovery
TADDM sensor scanning that discovers basic information about the active computer systems in the runtime environment. This scanning is also known as credential-less discovery because it requires no credentials. It uses the Stack Scan sensor and the IBM® Tivoli® Monitoring Scope sensor. Level 1 discovery is very shallow. It collects only the host name, operating system name, IP address, fully

qualified domain name, and Media Access Control (MAC) address of each discovered interface. Also, the MAC address discovery is limited to Linux on System z® and Windows systems. Level 1 discovery does not discover subnets. For any discovered IP interfaces that do not belong to an existing subnet that is discovered during Level 2 or Level 3 discovery, new subnets are created based on the value of the `com.collation.IpNetworkAssignmentAgent.defaultNetmask` property in the `collation.properties` file.

Level 2 discovery
TADDM sensor scanning that discovers detailed information about each operating system in the runtime environment. This scanning is also known as credentialed discovery, and it requires operating system credentials. Level 2 discovery collects application names and the operating system names and port numbers that are associated with each running application. If an application has established a TCP/IP connection to another application, this information is collected as a dependency.

Level 3 discovery
TADDM sensor scanning that discovers detailed information about 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 credentialed discovery, and it requires both operating system credentials and application credentials.

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:

Operating system	Approximate number of CIs
Windows	500
AIX	1000
Linux	1000
HP-UX	500
Network devices	1000

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.

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.

utilization discovery
TADDM sensor scanning that discovers utilization information for the host system. A utilization discovery requires operating system credentials.

Chapter 1. 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
- Simple Object Access Protocol (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 applications, 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.

Related concepts

"Simplified Model" on page 15

As Common Data Model causes problems, a new Simplified Model for storing data is introduced in TADDM 7.3 version. The only elements that are left from the old model are classes.

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 SDK.

System requirements

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

Table 1 on page 2 lists the system elements and describes the respective requirement details.

Table 1. TADDM SDK system requirements

Element	Details
Operating system	Any operating system that supports the required Java runtime environment (JRE)
Memory	2GB
Processors	1
Processor speed	1GHz
Disk space	200MB (Including the JVM)
Additional software requirements	If you are running the SDK on the same computer as the TADDM server, use the IBM Java SDK version 7.0 provided with the TADDM server. The IBM Java SDK is located in the `$COLLATION_HOME/external` directory. If you are installing the SDK on a different computer from the TADDM server, the Java SDK version 7.0 is required. If the client is not on the same machine as the server, then the Java SDK levels must match. For example, do not try to run a client with version 5.0 of the Java SDK with a server running version 7.0.

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

Directory	Contents
dist/	TADDM root directory which is also referred to as the COLLATION_HOME directory. 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.
bin/	
deploy/	
etc/	
external/	
lib/	
log/	
dist/sdk/	Contains the TADDM SDK. See the table on the Standalone mode directory structure for more information. .

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

Directory	Contents
adaptor	Contains TADDM Discovery Library Adapter 1.0.
bin	Contains useful shell scripts and batch files
dla	Contains IBM Discovery Library IdML Certification Tool
doc	Contains English pdfs and other documentation files
etc	Configuration properties
examples	Samples directory

Table 3. Standalone mode directory structure (continued)	
Directory	**Contents**
`lib`	Server and client runtime libraries
`log`	Runtime logs
`schema`	The XML Schema

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

Set the JAVA_HOME environment variable to the directory for the Java runtime environment.

If JAVA_HOME is not set, the script runs the first Java executable file found on the execution path.

Setting configuration properties

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

Table 4. Configuration properties	
Parameter	**Details**
`com.collation.version`	Version of the API
`com.collation.LogFile`	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.
`com.collation.log.level`	Logging level, from among the following values: • INFO—Default • ERROR • DEBUG
`com.collation.log.filesize`	Log file size. The default value is 20 MB.
`com.collation.log.filecount`	Rollover count. The default value is 3.

Table 4. Configuration properties (continued)	
Parameter	**Details**
`com.ibm.cdb.service.registry.public.port`	Default port for the TADDM public services RMI registry. SDK (API) is one of the public TADDM services. This value must be the same as the setting for the TADDM server. The default value is 9433. If the API connects to multiple TADDM servers, you must configure all servers to use the same port, or specify the port when connecting.
`com.ibm.cdb.service.ApiServer.port`	Default port for the TADDM where API server listens on for non-SSL requests. SDK (API) is one of the public TADDM services. 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.

The `com.ibm.cdb.service.registry.public.port` and `com.ibm.cdb.service.registry.public.port` properties settings must match the settings 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 scopes defined in the TADDM server in XML format.
6. Collect all computer systems in the TADDM server by running the following command:

   ```
   % ./api.sh -u user -p password -H host find ComputerSystem
   ```

   The command returns all discovered computer systems in XML format.

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 Data Management Portal. You can use the same user ID and password for API access and the Data Management Portal.

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:taddm.server.web.port/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:taddm.server.web.port/GetTaddmVersion/getVersion/clientjar
```

- `platform-model.jar`:

```
http://taddm.server.machine.name:taddm.server.web.port/GetTaddmVersion/getVersion/modeljar
```

- `oal-topomgr.jar`:

```
http://taddm.server.machine.name:taddm.server.web.port/GetTaddmVersion/getVersion/
oaltopomgrjar
```

where *taddm.server.machine.name* is the fully qualified domain name of the server where TADDM is running and *taddm.server.web.port* is the HTTP port defined for TADDM server, whose default value is 9430.

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:

```
http://taddm.server.machine.name:taddm.server.web.port/GetTaddmVersion/getVersion/taddmversion
```

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

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

- `$COLLATION_HOME/sdk/lib/taddm-api-client.jar`
- `$COLLATION_HOME/sdk/lib/platform-model.jar`

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.
- *IBM Tivoli Common Data Model: Guide to Best Practices* at http://www.redbooks.ibm.com/abstracts/redp4389.html.

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 TADDM. 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, "source", is a ModelObject, and the following queries return the same results:

      ```
      SELECT * FROM TransactionalDependency WHERE source.guid ==
      'E72B13789C9039BFB32E3822FE50C197'
      ```

      ```
      SELECT * FROM TransactionalDependency WHERE source ==
      '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 a interface. Any class that implements a 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 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 OmmittedIdentifier (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 ("\nShort 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]);
               }
```

```
                i++;
            }
            System.out.println("\nThe following classes must be specified using
              the long name: ");
            System.out.println(dups);
            sess1.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        } finally {
            if (api != null) {
              try {
                 api.close();
              } catch (Exception e) { }
            }
        }

    }

}
```

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 (Java EE) 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).

Simplified Model

As Common Data Model causes problems, a new Simplified Model for storing data is introduced in TADDM 7.3 version. The only elements that are left from the old model are classes.

With the new Simplified Model, you can create script-based, custom sensors more easily and extend the scope of discovery. You can customize new sensors and edit the existing ones.

Note: Simplified Model is not supported for other products that you integrate with TADDM.

The following table lists the most important features introduced with Simplified Model and provides the location where you can find more information about them.

Table 5. Simplified Model features

Feature	Location
Top-level generic classes in the hierarchy that represent crucial CIs.	“Package, classes, and hierarchies” on page 15
New middle-level objects that represent deployable components.	“Package, classes, and hierarchies” on page 15
A mechanism to extend new top levels with the unlimited number of attributes (extended attributes).	“Extended attributes” on page 19
A mechanism to store entire chunks of raw data (for example XMLs, or command outputs) and attach them to the top-level CIs (extended instances).	“Extended instances” on page 25
New `eval` operator used in MQL queries for extended attributes and extended instances.	“Model Query Language overview” on page 35
New generic naming rule attribute `openId`.	“OpenId generic naming rule attribute” on page 17

Package, classes, and hierarchies

Package and classes

All new classes are stored in the `com.collation.platform.model.topology.simple` package. The classes names start with the capital letter "S" to avoid conflicts because TADDM differentiates data types by their short names.

Hierarchy attributes

Each data model object contains two hierarchy attributes, `hierarchyDomain` and `hierarchyType`, which define information that in the old model was present in the package and type of every class. For example, the `ComputerSystem` type contained many specific computer systems that represented different operating systems like `sys.linux.LinuxUnitaryComputerSystem` or `sys.windows.WindowsComputerSystem`. These objects had to be stored separately. The new simplified model allows for storing objects of both types as the `SComputerSystem` type, when these two attributes are set in the following way:

- For `sys.linux.LinuxUnitaryComputerSystem`:

```
hierarchyDomain="sys.unix.linux"
hierarchyType="RedHat"
```

- For `sys.windows.WindowsComputerSystem`:

```
hierarchyDomain="sys.windows"
hierarchyType="Windows7"
```

The `hierarchyDomain` attribute values specify levels of domain, beginning with the most general level and finishing with the most specific one. For example, in the `"app.db.mongodb"` value, `app` is an application server, `db` is a database server, and `mongodb` is a specific database, in this case MongoDB.

The hierarchy attributes are used to fully handle UI. They are also used in querying.

New hierarchy types

The following list shows new hierarchy types:

- `SComputerSystem` - replaces the `ComputerSystem` hierarchy.
- `SSoftwareServer` - replaces the `AppServer` hierarchy.
- `SLogicalGroup` - replaces the `AppServerCluster` hierarchy and represents any kind of collections that are stored by a sensor, for example, clusters, or domains.
- `SFunction` - replaces the `Function` hierarchy and represents additional rules and functions.
- `SSoftwareInstallation` - represents a physical software packages that are present on a computer system that constitutes a software server (vendor created libraries).
- `SPysicalFile` - replaces `AppConfig` and `LogicalContent` hierarchies and represents a configuration file. The whole content of this file is captured by a discovery.
- `SDeployableComponent` - a new type that represents types that are considered as relationship sources or targets, which are deployed on computer systems, software servers, or logical groups. They are middle-level objects. Some of the old CDM types are still attached to `SDeployableComponent` hierarchy to ensure compatibility with earlier versions. The following list specifies such types:
  - `BiztalkApplication`
  - `Database`
  - `DominoDatabase`
  - `ExchangeLink`, `ExchangeStorageGroup`
  - `FileSystem`
  - `IIsWebServer`, `IIsWebVirtualDir`
  - `MBExecutionGroup`, `MBMessageFlow`, `MessageBox`
  - `MQChannel`, `MQQueue`
  - `OracleSchema`
  - `SharePointWebApplication`
  - `SoftwareModule` - the only component of the `SoftwareModule` type that is left is `J2EEApplication` as it is the only deployable component in this type.
  - `WebVirtualHost`

  All these new types inherit from the following abstract types that are introduced into data model for modeling purposes: `SStandaloneObject`, `SContextualObject`, `SGroup`.

`lastStoreTime` attribute

The `lastModifiedTime` attribute is deprecated because it has a misleading name. A new `lastStoreTime` attribute replaces it.

Migration

The migration from the old model to the new one is automatic. You do not need to complete any additional tasks. The MQL queries and SQL views are compatible with earlier version.

Changes in Data Management Portal

In the **Inventory Summary** pane on the **Inventory** tab, there is a new component type called `Generics`. You can find there objects of classes from the `simple` package. You can also browse generic folders in the Discovered Components pane and display details for discovered objects of new class types. For information about displaying extended attributes and extended instances, see "Extended attributes" on page 19 and "Extended instances" on page 25.

Business applications

The Simplified Model is supported by business application engine. You can use new classes, attributes, and the `eval` operator in queries while working with business applications. You can also create business application from the objects that you store by using the new model.

OpenId generic naming rule attribute

For each object, you must specify attributes, and define a naming rule that indicates which of these attributes provide a unique value. The naming rules vary depending on the hierarchy type. The new generic types that are stored in the `simple` package, are stored with the use of the same naming rules as in the old CDM. Objects that inherit from the `SStandaloneObject` type are stored with the `openId` attribute. Objects that inherit from the `SContextualObject` type are stored with the `context` and `scopedId` attributes.

Both `openId` and `scopedId` attributes are of the string type in a database, but on the API level they are of a new Java type `OpenId`. As a result, those attributes have a specific format, aligned with the OpenId schema. The `OpenId` type contains methods to easily construct any meaningful string that represents a value of a naming rule. The values that are specified in the `openId` attribute are the source for GUID calculation.

Examples

Example 1

For servers, naming rule is usually based on two attributes. They are the primary service access point that is created from the server's primary IP address, and a port, on which this service listens on. The `openId` attribute is specified in the following way:

```
id = OpenId().addId('IP' , seed.getPrimaryIpAddress().getStringNotation())
.addId('port' , str(seed.getPort()))
```

Example 2

In Common Data Model, the old `ComputerSystem` type has a naming rule based on the `manufacturer`, `model`, and `serialNumber` attributes. These three attributes are defined in the class explicitly and when values are set for them, a `ComputerSystem` object can be stored.

```
LinuxUnitaryComputerSystem cs = ModelFactory.newInstance
(LinuxUnitaryComputerSystem.class);
cs.setManufacturer("RedHat");
cs.setModel("Linux");
cs.setSerialNumber("as00123012");
```

Then an object with the following attribute map is stored in the persistence layer :

```
manufacturer -> RedHat
serialNumber -> as00123012
isPlaceholder -> false
model -> Linux
```

In the new Simplified Model, storing objects of the simplified `SComputerSystem` type with the same values looks the same. However, if there is another naming rule attribute available for the particular computer system, for example an id that a cluster assigns to every physical computer system that it manages, and such naming rule attribute is not defined in the model, it can be extended with the use

of the OpenId type. When the attribute is extended, it is not only stored but it also uniquely represents the computer system. The OpenId type is used in the following way:

```
SComputerSystem scs = ModelFactory.newInstance(SComputerSystem.class);
scs.setHierarchyDomain("sys.unix.linux");
scs.setHierarchyType("RedHat");
scs.setManufacturer("RedHat");
scs.setModel("Linux");
scs.setSerialNumber("as00123012");
OpenId id = new OpenId();
id.addId("clusterInternalId", "66");
scs.setOpenId(id);
```

Example 3

The OpenId type can also be set in the following way:

```
scs.setOpenId(new OpenId().addId("clusterInternalId", "66"));
```

The following attribute map is stored:

```
manufacturer -> RedHat
serialNumber -> as00123012
model -> LinuxhierarchyType -> RedHat
isPlaceholder -> false
hierarchyDomain -> sys.unix.linux
openId -> <openId><id><name>clusterinternalid</name><value>66</value></id>
</openId>
```

Example 4

Adding a function to this simplified computer system is very similar. The following example shows how to create the OpenId attribute from values that are already set for the simplified class attributes:

```
SFunction sf = ModelFactory.newInstance(SFunction.class);
sf.setName("Cisco Firewall");
sf.setHierarchyDomain("function.net.firewall");
sf.setHierarchyType("Cisco");

OpenId fid = new OpenId(sf);
fid.addId("name", null);
fid.addId("type", "firewall");
sf.setScopedId(fid);
sf.setProvider(scs);
```

The following attribute map is stored:

```
hierarchyType -> Cisco
isPlaceholder -> false
provider -> {hierarchyType=RedHat;hierarchyDomain=sys.unix.linux;
isPlaceholder=false;openId=<openId><id><name>clusterinternalid</name>
<value>66</value></id></openId>;}
hierarchyDomain -> function.net.firewall
name -> Cisco Firewall
scopedId -> <openId><id><name>name</name><value>Cisco Firewall</value>
</id><id><name>type</name><value>firewall</value></id>
</openId>
```

Example 5

To easily create one simple id without any distinction for particular attributes inside, set the attribute in the following way:

```
OpenId fid = new OpenId(sf);
sf.setProvider(scs);
sf.setScopedId(new OpenId().addId("id19921"));
```

The following attribute map is stored:

```
hierarchyType -> Cisco
isPlaceholder -> false
provider -> {hierarchyType=RedHat;hierarchyDomain=sys.unix.linux;
isPlaceholder=false;openId=<openId><id><name>clusterinternalid</name>
<value>66</value></id></openId>;}
hierarchyDomain -> function.net.firewall
```

```
name -> Cisco Firewall
scopedId -> <openId><id><name>id</name><value>id19921</value></id></openId>
```

Related concepts

“Extended attributes” on page 19

In the old Common Data Model, extended attributes were used to manually define up to 100 attributes per CDM type to store the data. They were used to extend CDM types with attributes out of domain, for example server room number, and to expand the scope of the discovery of CIs. In the new Simplified Model, extended attributes are used to store all simple attributes of CDM classes that were previously present in the types lower in hierarchy.

Extended attributes

In the old Common Data Model, extended attributes were used to manually define up to 100 attributes per CDM type to store the data. They were used to extend CDM types with attributes out of domain, for example server room number, and to expand the scope of the discovery of CIs. In the new Simplified Model, extended attributes are used to store all simple attributes of CDM classes that were previously present in the types lower in hierarchy.

For example, in the old model, the `ExchangeServer` type had the `productID` attribute of the string type defined. In the new model, the `ExchangeServer` type is stored in the following way:

```
SSoftwareServer sr = ModelFactory.newInstance(SSoftwareServer.class);
sr.setHierarchyDomain("app.messaging.exchange");
sr.setHierarchyType("Exchange");
sr.setOpenId(new OpenId().addId("serverName", "Exchange1122"));
```

The `productID` attribute cannot be stored because the `SSoftwareServer` type stores only essential attributes and cannot be extended. The extended attributes allow for storing such specific attributes.

Data model type

The following changes are introduced in the new data model:

- Extended attributes are stored along with the CIs in the XA attribute of a new custom type `ExtendedAttributesData`. The data that is kept in separate objects of the `UserData` type is migrated into the XA attribute.
- The limitation to store up to 100 attributes per CDM type was removed. The number of attributes that can be stored into a single object depends on the capacity of the database XML column type. Also, the `-g` option of the bulk load program can be used to store extended attributes.
- Extended attributes have categories. If no category is selected, an attribute is stored in the default `General` category. All extended attributes present in the old data model were moved into the default `General` category.
- Old attribute `byte[] extendedAttributes` is kept only to ensure compatibility with earlier versions and is deprecated. Public API methods `setExtendedAttributes` and `getExtendedAttributes` are deprecated as well.

Viewing extended attributes

After you run a discovery of a sensor, for which you specified the XA attribute, you can view extended attributes in Data Management Portal. Open the **Details** pane of a discovered object. There is no longer **Extended Attributes** tab, the information about extended attributes of the default category is displayed on the **General** tab. The extended attributes of a custom category are displayed on a custom tab. For example, extended attributes of Markers category are displayed on the **Markers** tab.

Example of usage in a sensor

The following example shows the `ExtendedAttributesData` type that is used to store extended attribute or attributes with a CI. The `productID` attribute is kept in a default category. A new category for physical location of a software server is created.

```
SSoftwareServer sr = ModelFactory.newInstance(SSoftwareServer.class);
sr.setHierarchyDomain("app.messaging.exchange");
sr.setHierarchyType("Exchange");
sr.setOpenId(new OpenId().addId("serverName", "Exchange1122"));

ExtendedAttributesData xa = new ExtendedAttributesData();
xa.addAttribute("productID", "ID12021");
xa.addAttribute("Location", "buildingNo", "North23");
xa.addAttribute("Location", "floorNo", "3");
xa.addAttribute("Location", "roomNo", "15");
xa.attachTo(sr);
```

Notes:

- In TADDM 7.3.0, and 7.3.0.1, you can use two methods to store extended attributes. One of them is `attachTo`, and it is used in the preceding example. This method must be specified after all `addAttribute()` entries. The alternative way is to use the `setXA` method, for example:

  ```
  xa.addAttribute("Location", "roomNo", "15");
  xa.toXML();
  sr.setXA(xa);
  ```

  If you use the `setXA` method, you must also specify the `toXML` method to convert extended attributes into a layout that can be stored.

- Fix Pack 2 In TADDM 7.3.0.2, there is no difference between the `attachTo`, and the `setXA` methods. Neither of them requires using the `toXML` method.

The following attribute map is stored:

```
hierarchyType -> Exchange
isPlaceholder -> false
openId -> <openId><id><name>servername</name><value>Exchange1122</value>
</id></openId>
hierarchyDomain -> app.messaging.exchange
XA -> <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xml>
  <attribute name="productID" category="taddm_global">ID12021</attribute>
  <attribute name="buildingNo" category="Location">North23</attribute>
  <attribute name="floorNo" category="Location">3</attribute>
  <attribute name="roomNo" category="Location">15</attribute>
</xml>
```

Partial update

A CI can be stored with the same naming rules values from different data sources, and as a result the values are merged into one object. To prevent that, partial update mechanism is used to merge two different XML-formatted XA attributes. For example, if one source is stored with object 1, and another source is stored with object 2, a CI that holds the merged attribute is created.

Object 1

```
SSoftwareServer sr = ModelFactory.newInstance(SSoftwareServer.class);
sr.setHierarchyDomain("app.messaging.exchange");
sr.setHierarchyType("Exchange");
sr.setOpenId(new OpenId().addId("serverName", "Exchange1122"));

ExtendedAttributesData xa = new ExtendedAttributesData();
xa.addAttribute("productID", "ID12021");
xa.addAttribute("internal", "ID1233");
xa.addAttribute("Location", "buildingNo", "North23");
xa.attachTo(sr);
```

Object 2

```
SSoftwareServer sr = ModelFactory.newInstance(SSoftwareServer.class);
sr.setHierarchyDomain("app.messaging.exchange");
sr.setHierarchyType("Exchange");
sr.setOpenId(new OpenId().addId("serverName", "Exchange1122"));

ExtendedAttributesData xa = new ExtendedAttributesData();
xa.addAttribute("productID", "ID12024");
xa.addAttribute("customID", "ID12333");
xa.addAttribute("Location", "floorNo", "3");
xa.addAttribute("Location", "roomNo", "15");
xa.attachTo(sr);
```

Merged attribute

```
hierarchyType -> Exchange
isPlaceholder -> false
openId -> <openId><id><name>servername</name><value>Exchange1122</value>
</id></openId>
hierarchyDomain -> app.messaging.exchange
XA -> <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xml>
  <attribute name="customID" category="taddm_global">ID12333</attribute>
  <attribute name="internal" category="taddm_global">ID1233</attribute>
  <attribute name="productID" category="taddm_global">ID12024</attribute>
  <attribute name="buildingNo" category="Location">North23</attribute>
  <attribute name="floorNo" category="Location">3</attribute>
  <attribute name="roomNo" category="Location">15</attribute>
</xml>
```

All not overlapping attributes from the first object and the second object are present with their values in the merged attribute and the `productId` attribute has a value from the object that was stored as last. When the XA attribute of any model object is partially updated, both category type and attribute name are included in the process.

Cache for metadata and validation

Extended attributes are defined rarely. Therefore, a cache is created to keep the data. Every time a model object is placed in a persistence tool to be stored, the tool validates the XA attribute against meta definitions that are taken from this cache. Every category type and attribute name pair that is present in the XA attribute, but not defined in the meta cache is removed from the XA attribute before the model object is persisted. When the cache refresh process is disabled, meta information is taken from the persistence layer. When the process is enabled, the meta information is taken from Java memory.

You can control the frequency of the cache refresh process by using the `com.ibm.cdb.ea.metaRefreshFrequency` property in the `collation.properties` file. The default value is `20` and is expressed in seconds. If you want to disable the cache refresh process, set the value of this property to `0`.

Note: When you are defining extended attributes, disable the cache refresh process and enable it when you finish.

MQL queries with EVAL operator (XA and XD attributes only)

Many CDM attributes are moved into XML content of the XA or XD attributes. Therefore, MQL syntax supports a new operator `eval`, which can be added in the where clause. The `eval` operator enables querying CIs by the values of extended attributes or extended instances.

Note: All MQL queries examples contain escaped quotation marks (\"*value*\") because it is assumed that the queries are run in the following manner:

```
./api.sh -u username -p password find "MQL query"
```

For example, the following query was run to find a computer system with the `productID` attribute set to `'prod1'`:

```
SELECT * FROM ComputerSystem WHERE productID == 'prod1'
```

The following equivalent query uses the `eval` operator:

```
SELECT * FROM ComputerSystem WHERE XA eval '/xml
[attribute[@category=\"taddm_global\" and @name=\"productID\"]=\"prod1\"]'
```

The `eval` operator can be followed by any valid XPath expression that returns Boolean true or false value to enable the performance of correct SQL filtering by the persistence layer.

More examples

- Find all ComputerSystems, which have any extended attribute with the `val` value:
  - MQL:

    ```
    SELECT * FROM ComputerSystem WHERE XA eval '/xml[attribute=\"val\"]'
    ```

  - SQL:

    ```
    SELECT * FROM compsys WHERE xmlexists('$c/xml[attribute="val1"]'
    passing compsys.xa_x as "c")
    ```

- Find a ComputerSystem with the `attr2` extended attribute, which has the category set to `Other` and value set to `two`:
  - MQL:

    ```
    SELECT * FROM ComputerSystem WHERE XA eval '/xml/attribute
    [@name=\"attr2\" and @category=\"Other\" and text()=\"two\"]'
    ```

  - SQL:

    ```
    SELECT * FROM compsys WHERE xmlexists('$c/xml/attribute[@name="attr2"
     and and @category="Other" and text()="two"]' passing compsys.xa_x as
     "c")
    ```

Creating extended attributes in Domain Management Portal

You can define extended attributes for a particular component in Domain Management Portal.

Procedure

To create an extended attribute, complete the following steps in Domain Management Portal.

Note: You can also define extended attributes by running the bulk load program.

1. On the menu bar, click **Edit** > **Extended Attributes**.

   The **Define Extended Attributes** window is displayed.
2. From the **Component type** menu, select the type of component, for which you want to create an extended attribute.
3. Click **New**.

   The **Create New Extended Attribute** window is displayed.
4. In the **Extended attribute name** field, type the name of the extended attribute.
5. From the **Extended attribute type** menu, select the type of extended attribute you want to create.
6. From the **Extended attribute category** menu, specify the category for your extended attribute.
7. Click **OK**.
8. In the **Define Extended Attributes** window, click **OK**.

   See also the *Define Extended Attributes window* topic in the TADDM *User's Guide*.

Creating extended attributes in the files

You can define extended attributes for a particular component in the files in the `$COLLATION_HOME/dist/etc/templates` directory.

Procedure

1. Make sure that the template files in the `$COLLATION_HOME/dist/etc/templates/commands` directory contain only `SCRIPT:etc/templates/commands/extension-scripts/ea_sensor_1.0.py`.
2. On the server that you want to discover, create a configuration file `/tmp/`*`file_name`*`.conf` that contains pairs of attributes that you want to define, and their values.
   For example:

   ```
   FirstAttribute = First value
   SecondAttribute = Second value
   ```

   List these attributes in the `extended_attributes.properties` file. See the next step.
3. In the `$COLLATION_HOME/etc/templates` directory, create the `extended_attributes.properties` file. It must contain the attributes that you want to collect. In the `extended_attributes.properties.example` file, you can find the structure of this file.
   An example entry:

   ```
   Linux.FileGEN.1.Key="/tmp/attribute.conf"
   Linux.FileGEN.1.Attributes="myAttribute"
   ```

   where:

   - `/tmp/attribute.conf` is the location of the file, where you specify the attribute that you want to define (`/tmp/`*`file_name`*`.conf`).
   - `myAttribute` is the name of the attribute that you want to define, which is specified in the `/tmp/attribute.conf` file.
4. In the Discovery Management Console, in the **Computer Systems** window, set the **Enable** value to *true*.

What to do next

Before you run a discovery, all the attributes that you want to collect must also be created in Domain Management Portal. If they are not, they are skipped during the discovery.

Important: During the discovery, the names of the attributes that you define in the files are changed. The `FileGEN` prefix is added to the beginning of each name. For example, if the name of your attribute is `myAttribute`, it is changed into `FileGEN_myAttribute`. Therefore, when you create attributes in Domain Management Portal, you must provide names with the prefix `FileGEN`. The attributes that are created in the files and in Domain Management Portal must have the same names. When you provide names without the prefix, the attributes are ignored.

Deleting extended attributes

You can remove existing extended attributes.

Procedure

To delete an extended attribute, complete the following steps in the Domain Management Portal:

1. On the menu bar, click **Edit** > **Extended Attributes**.
   The **Define Extended Attributes** window is displayed.
2. From the **Component type** menu, select the type of component, for which you want to delete an extended attribute.
3. Select an existing extended attribute.
4. Click **Delete**.

5. Click **OK**.

   The extended attribute is deleted.

Auto-defining extended attributes for public Java API and bulk load program

As defining extended attributes manually is time-consuming, you can enable automatic defining of extended attributes. You can work with extended attributes in Data Management Portal only to correct and tune extended attributes definitions.

External Java API and bulk load program enable TADDM storage manager to check whether any extended attributes attached to a CI can be found. If such attributes are found, those that were not defined as the `String` type can be automatically defined. With this mechanism, it is best to use the book or a Jython script to store the data along with the forced auto-definition, and then go to Data Management Portal to check the definitions, and adjust the attributes types and categories.

To enable automatic defining of extended attributes with the use of external Java API, use the `ApiSession` object's `setAutoDefineEAs` method, which is set to `true`, as in the following example:

```
conn = ApiFactory.getInstance().getApiConnection("localhost", -1, None,
Boolean(0))
sess = ApiFactory.getInstance().getSession(conn, "administrator", "collation",
ApiSession.DEFAULT_VERSION)
api = sess.createCMDBApi()

  print "Turning on auto-defining API feature for Extended Attributes"
sess.setAutoDefineEAs(Boolean(1))
```

To enable automatic defining of extended attributes with the use of the bulk load program, use the `-loadEAMeta` attribute, or set the `com.ibm.cdb.bulk.loadeametaflag=true` property in the `collation.properties` file. In this mode, the bulk load program does not store any CIs. Instead, it extracts the information about all extended attributes that are loaded and uses these extended attributes to create correct definitions. As a result, correct `UserDataMeta` objects are created in TADDM.

Note: You can use the `-g` option when running the bulk load program, which shortens the loading process.

Fix Pack 2 Auto-defining extended attributes for sensors

You can define extended attributes for any sensor and use them to store the sensor data automatically.

To define extended attributes for sensors, create an IdML book with extended attributes definitions. The name of this book must be `xa.xml`. Place the file in the particular OSGi bundle directory, for example `$COLLATION_HOME/osgi/plugins/com.ibm.cdb.discover.sensor.sys.examplesensor_1.0.0/xa.xml`.

Each time the TopologyBuilder service is started, new and changed books are automatically loaded. During the loading process, the bulk load program is run with the `-loadEAMeta` option enabled, automatically defining extended attributes. For details about the bulk load program, see “Auto-defining extended attributes for public Java API and bulk load program” on page 24.

To view the extended attributes in the TADDM UI, you must restart the TADDM server.

If you want to configure the bulk loading process of books that contain extended attributes, you can use the `$COLLATION_HOME/etc/bulkload.properties` file, which stores bulk load program configuration.

Example of usage

The `com.ibm.cdb.discover.sensor.sys.foobarsensor_1.0.0/xa.xml` file contains extended attributes definitions that are used by the example FoobarSensor. During the first server start, after the TopologyBuilder service is initialized, TADDM loads the `com.ibm.cdb.discover.sensor.sys.foobarsensor_1.0.0/xa.xml` book. Then, CRC32 sum of the `xa.xml` file is calculated and stored in the `com.ibm.cdb.discover.sensor.sys.foobarsensor_1.0.0/xa.xml.crc` file. The CRC32 sum is calculated each time the `xa.xml` file is changed. The books are loaded by the TopologyBuilder service only when the check sum changes.

Example structure of the `xa.xml` file

```
<idml:idml xmlns:idml="http://www.ibm.com/xmlns/swg/idml" xmlns:cdm=
"http://www.ibm.com/xmlns/swg/cdm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/swg/idml idml.xsd">
    <idml:source IdMLSchemaVersion="0.8">
        <cdm:process.ManagementSoftwareSystem CDMSchemaVersion="2.8">
            <cdm:MSSName>your data</cdm:MSSName>
            <cdm:Hostname>your data</cdm:Hostname>
            <cdm:ManufacturerName>your data</cdm:ManufacturerName>
            <cdm:ProductName>your data</cdm:ProductName>
            <cdm:ProductVersion>your data</cdm:ProductVersion>
            <cdm:Label>your data</cdm:Label>
        </cdm:process.ManagementSoftwareSystem>
    </idml:source>

    <!-- Operation... -->
    <idml:operationSet opid="1">
        <idml:create timestamp="2012-07-12T05:10:58Z">
        <cdm:CDM-ER-Specification>
            <cdm:sys.ComputerSystem id="CS" sourceToken="CS">
            <cdm:extension>
                <cdm:extattr name="sensor_foobar_xa1" category="_internal"></
cdm:extattr>
            </cdm:extension>
            </cdm:sys.ComputerSystem>
        </cdm:CDM-ER-Specification>
        </idml:create>
    </idml:operationSet>
</idml:idml>
```

Extended instances

In the old Common Data Model, the large chunks of data had to be split into the key-value pairs and stored as extended attributes. Now you can use extended instances for storing large content, thanks to which you do not need to create a large number of extended attributes. You can store low-level objects as extended instances and attach them to top-level objects, which are CIs, as raw or preprocessed data.

Data model type

The structure of the XD attribute is a simple sequence of XML elements. Every model object type has the XD attribute of a new Java `ExtendedInstanceData` type. This type allows for storing any type of content, for example plain text, like command output, XML, CSV, or CDATA.

You can group extended instances by type, the default type is `General`.

Viewing extended instances

After you run a discovery of a sensor, for which you specified the XD attribute, you can view extended instances in Data Management Portal. Open the **Details** pane of a discovered object. Each extended instance type is displayed on a separate tab.

Example of usage in a sensor

The following example shows the `ExtendedInstanceData` type that is used to store raw content. The `lsofOutput` and `ifconfigOutput` variables are the string variables that store command outputs. In this example, the outputs are not processed, but they can be formatted as an XML or JSON.

```
        SComputerSystem scs = ModelFactory.newInstance(SComputerSystem.class);
        scs.setHierarchyDomain("sys.unix.linux");
        scs.setHierarchyType("RedHat");
        scs.setManufacturer("RedHat");
        scs.setModel("Linux");
        scs.setSerialNumber("as00123012");

        ExtendedInstanceData xd = new ExtendedInstanceData();
        xd.addInstance("lsof", lsofOutput);
```

```
        xd.addInstance("ipInterfaces", ifconfigOutput);
        xd.addInstance(null, "Linux vmw009128109120 2.6.32-220.el6.x86_64 #1
SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64
GNU/Linux");
        scs.setXD(xd);
```

As a result, the following map of attributes with the XD attribute filled with correct XML is stored. The XML creates elements for the instance type with the elements of raw data of the same type inside, enclosed by <instance>. The instance that is created with the null type is placed in the default type <general>.

```
manufacturer -> RedHat
hierarchyType -> RedHat
XD -> <xml>
    <general>
        <instance>Linux vmw009128109120 2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13
EST 2011 x86_64 x86_64 x86_64 GNU/Linux</instance>
    </general>
    <ipInterfaces>
        <instance>eth4      Link encap:Ethernet  HWaddr 00:50:56:00:72:92
          inet addr:9.128.109.120  Bcast:9.128.109.255  Mask:255.255.255.0
          inet6 addr: fe80::250:56ff:fe00:7292/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:805298636 errors:0 dropped:0 overruns:0 frame:0
          TX packets:665767802 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:590819760949 (550.2 GiB)  TX bytes:272393336858 (253.6 GiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:833223633 errors:0 dropped:0 overruns:0 frame:0
          TX packets:833223633 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:594042898379 (553.2 GiB)  TX bytes:594042898379 (553.2 GiB)

        </instance>
    </ipInterfaces>
    <lsof>
        <instance>pickup    31017   postfix  mem       REG                8,2  1580928
    1066619 /usr/lib64/mysql/libmysqlclient.so.16.0.0
              pickup    31017   postfix  mem       REG                8,2   184088
     139800 /lib64/libpcre.so.0.0.1
              pickup    31017   postfix  mem       REG                8,2    63304
     139295 /lib64/liblber-2.4.so.2.5.6
              pickup    31017   postfix  mem       REG                8,2   308912
     130952 /lib64/libldap-2.4.so.2.5.6
              pickup    31017   postfix  mem       REG                8,2   156872
     130819 /lib64/ld-2.12.so
              pickup    31017   postfix    0u      CHR                1,3      0t0
       3640 /dev/null
              pickup    31017   postfix    1u      CHR                1,3      0t0
       3640 /dev/null
              pickup    31017   postfix    2u      CHR                1,3      0t0
       3640 /dev/null
              pickup    31017   postfix    3r     FIFO                0,8      0t0
      10751 pipe
              pickup    31017   postfix    5u     unix 0xffff8802350d9c80      0t0
      10659 socket
              pickup    31017   postfix    6u     FIFO                8,2      0t0
     392456 /var/spool/postfix/public/pickup
              pickup    31017   postfix    7u     unix 0xffff8802378e8680      0t0
   49305400 socket
              pickup    31017   postfix    8u      REG                0,9        0
       3638 anon_inode
              loop0     31473      root  cwd       DIR                8,2     4096
          2 /
              loop0     31473      root  rtd       DIR                8,2     4096
          2 /
              loop0     31473      root  txt   unknown
           /proc/31473/exe
        </instance>
    </lsof>
</xml>
serialNumber -> as00123012
isPlaceholder -> false
hierarchyDomain -> sys.unix.linux
model -> Linux
```

The preceding example uses the simplest `addInstance` methods. It is advised to use the `addInstance(String type, INSTANCE_FORMAT format, boolean isVisible, String content)` method that enables the control of two XML attributes of the `<instance>` element.

The first XML attribute is `format`, which is used to interpret the content. `INSTANCE_FORMAT` is a Java enum with the following possible values: `plain`, `XML`, `CSV`, `JSON`, `CDATA`.

The second XML attribute is `visible`, which is used when some instances are not displayed in the UI.

Example:

```
<general><instance format="plain" visible="false">Linux vmw009128109120
2.6.32-220.el6.x86_64 #1 SMP Wed Nov 9 08:03:13 EST 2011 x86_64 x86_64 x86_64
GNU/Linux</instance></general>
```

Partial update

A CI can be stored with the same naming rules values from different data sources, and as a result the values are merged into one object. To prevent that, partial update mechanism is used to merge two different XML-formatted XD attributes. For example, if one source is stored with object 1, and another source is stored with object 2, a CI that holds the merged attribute is created.

Object 1

```
        SComputerSystem xd1 = ModelFactory.newInstance
(SComputerSystem.class);
        xd1 = ModelFactory.newInstance(SComputerSystem.class);
        xd1.setHierarchyDomain("sys.unix.linux");
        xd1.setHierarchyType("RedHat");
        xd1.setManufacturer("RedHat");
        xd1.setModel("Linux");
        xd1.setSerialNumber("as00123012xd");

        ExtendedInstanceData xdone = new ExtendedInstanceData();
        xdone.addInstance("lsof", "content from CI1");
        xdone.addInstance(null, "content from CI1");
        xd1.setXD(xdone);
```

Object 2

```
        SComputerSystem xd2 = ModelFactory.newInstance
(SComputerSystem.class);
        xd2 = ModelFactory.newInstance(SComputerSystem.class);
        xd2.setHierarchyDomain("sys.unix.linux");
        xd2.setHierarchyType("RedHat");
        xd2.setManufacturer("RedHat");
        xd2.setModel("Linux");
        xd2.setSerialNumber("as00123012xd");

        ExtendedInstanceData xdtwo = new ExtendedInstanceData();
        xdtwo.addInstance("ips", "content from CI2");
        xdtwo.addInstance(null, "content from CI2");
        xd2.setXD(xdtwo);
```

Merged attribute

```
manufacturer -> RedHat
hierarchyType -> RedHat
XD -> <xml>
    <general>
        <instance>content from CI2</instance>
    </general>
    <lsof>
        <instance>content from CI1</instance>
    </lsof>
    <ips>
        <instance>content from CI2</instance>
    </ips>
</xml>
serialNumber -> as00123012xd
isPlaceholder -> false
hierarchyDomain -> sys.unix.linux
model -> Linux
```

All not overlapping attributes from the first object and the second object are present with their values in the merged attribute and the `general` type has the instances from the object that was stored as last. The partial update for the XD attribute of every model object is made with respect to an instance type.

MQL queries with EVAL operator (XA and XD attributes only)

Many CDM attributes are moved into XML content of the XA or XD attributes. Therefore, MQL syntax supports a new operator `eval`, which can be added in the where clause. The `eval` operator enables querying CIs by the values of extended attributes or extended instances.

Note: All MQL queries examples contain escaped quotation marks (\"*value*\") because it is assumed that the queries are run in the following manner:

```
./api.sh -u username -p password find "MQL query"
```

For example, the following query was run to find a computer system with the `productID` attribute set to `'prod1'`:

```
SELECT * FROM ComputerSystem WHERE productID == 'prod1'
```

The following equivalent query uses the `eval` operator:

```
SELECT * FROM ComputerSystem WHERE XA eval '/xml
[attribute[@category=\"taddm_global\" and @name=\"productID\"]=\"prod1\"]'
```

The `eval` operator can be followed by any valid XPath expression that returns Boolean true or false value to enable the performance of correct SQL filtering by the persistence layer.

More examples

- Find all ComputerSystems, which have any extended attribute with the `val` value:
  - MQL:

    ```
    SELECT * FROM ComputerSystem WHERE XA eval '/xml[attribute=\"val\"]'
    ```

  - SQL:

    ```
    SELECT * FROM compsys WHERE xmlexists('$c/xml[attribute="val1"]'
    passing compsys.xa_x as "c")
    ```

- Find a ComputerSystem with the `attr2` extended attribute, which has the category set to `Other` and value set to `two`:
  - MQL:

    ```
    SELECT * FROM ComputerSystem WHERE XA eval '/xml/attribute
    [@name=\"attr2\" and @category=\"Other\" and text()=\"two\"]'
    ```

  - SQL:

    ```
    SELECT * FROM compsys WHERE xmlexists('$c/xml/attribute[@name="attr2"
     and and @category="Other" and text()="two"]' passing compsys.xa_x as
     "c")
    ```

Extending sensor discovery scope with Simplified Model

You can easily extend the discovery scope of a sensor by using new functions of the Simplified Model. The following procedure shows an example configuration, on which you can base customization of your own sensors.

About this task

The following procedure shows the configuration of a simple script-based sensor for MongoDB database, when MongoDB server is installed.

Prerequisites

If you want the sensor to store extended attributes, first you must define them for a particular data model type. For detailed instruction, see “Creating extended attributes in Domain Management Portal” on page 22.

For this procedure, two extended attributes were created with the following parameters:

- `SDeployableComponent` type:
  - Extended attribute name - hasDatabaseIndexDef
  - Extended attribute type - Boolean
  - Extended attribute category - General
- `SSoftwareServer` type:
  - Extended attribute name - DatabaseCnt
  - Extended attribute type - Integer
  - Extended attribute category - Markers

Note: The following procedure does not provide details about new functions of the Simplified Model. It rather shows how to use these functions. For more information, see the appropriate topics in the “Simplified Model” on page 15 section of the TADDM documentation.

Procedure

1. Create custom server template by using **Custom Servers** pane in Discovery Management Console.

   Custom Servers pane contains templates of sensors that can be run by generic server sensor or as custom application server sensor.
   Open the details for MongoDB server. It is enabled. It stores objects of the AppServer type. In the Identifying Criteria section, you must specify a file name. When generic server sensor finds a process that contains this file name, it runs the custom sensor. In this case, the file name is `mongod.exe`.

   You can now run a discovery but only RuntimeProcess objects are discovered.
2. Configure extension for template sensor.
   a) Open the TADDM home directory and go to the `dist/etc/templates/commands` directory. Create a command file with the same name as the template defined in **Custom Servers**, in this case, `MondoDB`.

   The content of this file is executed when the custom sensor is run.
   The `MongoDB` file contains the following line:

   ```
   SCRIPT[com.ibm.cdb.core.jython253_2.5.3]:etc/templates/commands/
   extension-scripts/mongodb.py
   ```

   It means that the `mongodb.py` script is executed when the sensor is run.

   b) Configure the `mongodb.py` script.

   The following sections provide brief description of the most important elements of this script.

   The default beginning of the sensor

   ```
   (os_handle, result, appserver, seed, log, env) = sensorhelper.init
   (targets)
   ```

   This section specifies parameters that are passed by platform that is called `targets`:

   - `os_handle` - an operating system object.
   - `result` - the result, where the objects are attached.
   - `appserver` - the app server object that represents runtime processes that match the template's condition.
   - `seed` - the seed object that triggers the sensor.
   - `log` - logger.

- env - environment settings.

Creating the main object - defining class and hierarchy attributes

```
mdbserver = ModelFactory.newInstance(Class.forName('com.collation.
platform.model.topology.simple.SSoftwareServer'))
mdbserver.setHierarchyDomain('app.db.mongodb')
mdbserver.setHierarchyType('MongoDBServer')
```

In this section, a new type of class is used, SSoftwareServer. There are also new attributes, which position a CI that is represented by this object on the maps of domains. These attributes provide necessary details for the sensor.

- HierarchyDomain - specifies levels of domains. In this case, in the app.db.mongodb value, app is an application server, db is a database server, and mongodb is a specific database.
- HierarchyType - specifies the type of the object. In this case, mongodb that is used in the HierarchyDomain attribute represents a database server, so the HierarchyType attribute is set to MongoDBServer.

OpenId - a new generic naming rule attribute

```
print 'Primary IP : ' + seed.getPrimaryIpAddress().
getStringNotation()
print 'Port       : ' + str(seed.getPort())
id = OpenId().addId('IP' , seed.getPrimaryIpAddress().
getStringNotation()).addId('port' , str(seed.getPort()))
ID = OpenId.generateDisplayName(id)
mdbserver.setInstanceID(ID)
mdbserver.setOpenId(id)
mdbserver.setHostComputerSystem(os_handle.getComputerSystem())
```

You must name the object that you created. To do so, you must define a naming rule and specify attributes that provide unique values. For servers, naming rule is usually based on two attributes. They are the primary service access point that is created from the server's primary IP address, and a port, on which this service listens on.

The new OpenId attribute is a wrapper for a string attribute that gathers all values that are used for GUID calculation. This new attribute arranges the values in the same order every time. The addId method adds objects and set them as the value of the OpenId attribute.

There are also other, predefined attributes set for this sensor, for example instance ID, or host computer system to attach the server to a computer system that is passed by a platform as an operation system object attribute.

Setting the result object

```
print 'Setting mongodb server'
result.setServer(mdbserver)
```

Set the result object with the MongoDB server. If during the discovery the sensor fails, at least this information is stored.

Extract databases function

```
def extractDatabases(mgbserver, dbListLines):
  databases = list([])
  dbDir = None
  currentDatabase = None
  XD = None
  XA = None

  for dbLine in dbListLines:
    print 'Processing line : ' + dbLine
    if "DATABASE" in dbLine :
      print 'Found a database ' + dbLine
      #Create physical files
      print 'Attach as files'
      dbLineTokens = dbLine.split(" ")
      dbName = dbLineTokens[2].strip()
      dbDir = dbLineTokens[4].strip()
```

```
      print "Database name " + dbName
      print "Database directory " + dbDir

      #Create deployable components
      print "Create deployable component for database"
      database = ModelFactory.newInstance(Class.forName
('com.collation.platform.model.topology.simple.
SDeployableComponent'))
      database.setHierarchyDomain('app.db.mongodb')
      database.setHierarchyType('Database')
      database.setOpenId(OpenId().addId("databaseName", dbName))
      database.setName(dbName)
      database.setSoftwareServer(mgbserver)
      databases.append(database)
      currentDatabase = database
      XA = ExtendedAttributesData()
      XA.attachTo(currentDatabase)
      XD = ExtendedInstanceData()
      currentDatabase.setXD(currentXD)

    elif dbDir is not None and dbDir in dbLine and "Metadata for"
in dbLine:
      print "Found file " + dbLine + " to read for XD"
      databaseConfFileLineTokens = dbLine.split(" ")
      fileType = databaseConfFileLineTokens[3]
      fileName = databaseConfFileLineTokens[5]
      print "Instance type " + fileType
      print "Files content to read " + fileName

      if not fileName.endswith("bson"):
        cnfFileContent = readFileContent(fileName)
        print "Read content : " + cnfFileContent
        XD.addInstance(fileType,
ExtendedInstanceData.INSTANCE_FORMAT.json, 1, cnfFileContent)
        print "Added instance"

        if "index" in cnfFileContent :
          XA.addAttribute("hasDatabaseIndexDef", "true")
          xml = XA.toXML()
          print "Add XA marker for database " + dbName +
" with XML " + xml

  print "Attach files and deployable components"
  mgbserver.setDeployedComponents(tuple(databases))
  XA = ExtendedAttributesData()
  XA.addAttribute("Markers", DatabasesCnt", str(len(databases)))
  XA.attachTo(mgbserver)
  print "Add XA marker for server with xml " + xml
```

This function uses two containers for data. One contains extended attributes and the other extended instances.

In the `#Create deployable components` section, sensor creates object for discovered database by using the `SDeployableComponent` class type. The `HierarchyDomain` is set to `app.db.mongodb` because it is the same domain, but the `HierarchyType` attribute is set to `Database`. The naming rule is based on the database name, so the `openId` attribute is set to `databaseName`. A predefined attribute name is set (`database.setName(dbName)`), and the component is attached (`databases.append(database)`) to a server with software server attribute (`database.setSoftwareServer(mgbserver)`).

What is most important in this section, are two data containers. These containers are created for the newly created database. The XA attribute contains `ExtendedAttributesData` object for extended attributes and the XD attribute contains `ExtendedInstanceData` object for large chunks of data.

`ExtendedAttributesData`

By using the `XA = ExtendedAttributesData()` attribute, the sensor creates an `ExtendedAttributesData` object and adds it to a CI. By using `XA.addAttribute` method, it adds the attributes to this object, which are name and value. For example, in the extended attribute for the server object, sensor creates `ExtendedAttributesData` object, and adds the attributes, which are `DatabaseCnt` and category `Markers`. To convert the attribute into layout that can be stored, the sensor calls the `toXML` method.

ExtendedInstanceData

By using the `XD = ExtendedInstanceData()` attribute, the sensor creates an `ExtendedInstanceData` object. In this sensor, the XD attribute is used to store the contents of files that represent the inner structure of the database. The content of the database is read by the `readFileContent` function that is defined in the following way:

```
def readFileContent(fileName):
  print "Open file to read : " + fileName
  f = open(fileName, 'r')
  content = ""
  for line in f:
    content = content + line + "\n"
  return content
```

The following method is used to store the data:

```
XD.addInstance(fileType, ExtendedInstanceData.
INSTANCE_FORMAT.json, 1, cnfFileContent)
```

This particular `addInstance` method contains the following parameter:

- `fileType` - specifies the instance type.
- `ExtendedInstanceData.INSTANCE_FORMAT.json` - specifies the instance format.
- `1` - means true, and specifies that the content is visible on the details pane.
- `cnfFileContent` - specifies the instance content.

What to do next

- You can run the discovery of MongoDB with the updated scope.
- You can view the results of the discovery in Data Management Portal in the **Inventory Summary** pane. You can also browse generic folders in the Discovered Components pane and display details for the discovered objects.
- You can create business applications for the object that you store and display the topology.
- You can see more examples of the Jython script.

More samples of Jython script

If you want to discover extended attributes that you defined for the ComputerSystem (SComputerSystem), or AppServer (SSoftwareServer) class, refer to the following samples of the Jython script.

Note: If you are modifying the existing script, for example, the example script `ea_sensor_1.0.py`, overwrite the main part of the file. Also, the Standard Library Imports section must contain the `from com.collation.platform.model.util.ea import ExtendedAttributesData` line, like in the following example:

```
import sys
import java

from java.lang import System
from com.collation.platform.model.util.ea import ExtendedAttributesData
```

Jython script that is used to extend a computer system template - the main section

```
try:
    (os_handle, result, computerSystem, seed, log) = sensorhelper.
init(targets)

    patchVersionOut = sensorhelper.executeCommand("tail -1 /etc/
patch_status | cut -f2 -d,")
    patchDateOut = sensorhelper.executeCommand("tail -1 /etc/patch_
status | cut -f3 -d,")

    if patchVersionOut != None:
        XA = ExtendedAttributesData()
        XA.addAttribute("patchVersion", patchVersionOut)
```

```
        XA.addAttribute("patchDate", patchDateOut)
                XA.attachTo(computerSystem)
    else:
        log.info("patchVersion command return no output")
except:
    LogError("unexpected exception getting patchVersion
information")
```

Jython script that is used to extend a custom server template - the main section

```
try:
    (os_handle, result, appServer, seed, log, env) = sensorhelper.
init(targets)

    patchVersionOut = sensorhelper.executeCommand("tail -1 /etc/
patch_status | cut -f2 -d,")
    patchDateOut = sensorhelper.executeCommand("tail -1 /etc/patch_
status | cut -f3 -d,")

    if patchVersionOut != None:
        XA = ExtendedAttributesData()
        XA.addAttribute("patchVersion", patchVersionOut)
        XA.addAttribute("patchDate", patchDateOut)
        XA.attachTo(appServer)
    else:
        log.info("patchVersion command return no output")
except:
    LogError("unexpected exception getting patchVersion
information")
```

TADDM API overview

All discovery data that is displayed using the Data Management Portal 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 discovery data by using specific types of the application programming interface (API.)

All discovery data can be accessed by using the following types of API:

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

SOAP API
Exposes elements of the TADDM API as a Simple Object Access Protocol (SOAP) 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 6. XML document structure

XML	Description
`<?xml version="1.0" encoding="UTF-8"?>`	Header

`<results`
`xmlns="urn:www-collation-com:1.0"`
`xmlns:coll="urn:www-collation-com:1.0"`
`xmlns:xsi="http://www.w3.org/2001/`
`XMLSchemainstance"`
`xsi:schemaLocation=`
`"urn:www-collation-com:1.0`
`urn:www-collation-com:1.0/results.xsd">`
	Top-level result node, including the XML namespace specification

`<abbreviated_searched_class_name`
`xsi:type="full class name"`
`array="N"`
`LINK="guid"`
`lastModified="Last Modified Time in ms"`
`guid="unique model object id">`
	Class attributes, such as last modified time and unique object ID

`<property_name1>"text value"`
`</property_name1>`
`...`
`<property_nameX`
`xsi:type="full class name"`
`guid="unique model object id">`
`<property_name1>"text`
`value"</property_name1>`
`...`
`<property_nameN>"text`
`value"</property_nameN>`
`<property_nameX/>`
`...`
`<property_nameN>"text value"`
`</property_nameN>`
`<property_nameQ/>`
	Attribute values and embedded objects

`</abbreviated_searched_class_name>`	End of values
`</results>`	End of results

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.

JSON element	Description
`[`	Begins an array of model objects.
`{`	Begins a model object, which might contain zero or more model objects.
`"`*name*`":`*value*`,"`*name*`":`*value*	One or more name-value pairs, separated by commas.

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:

```
[{
"displayName":"esx3-vm16-rhes4",
"devices":[{"_class":"DiskDrive","guid":"2A2827686EB03465A955DE54BD3F6AB5"},
{"_class":"DiskDrive","guid":"D7DAF9DCD1E7347684A0D02E36E212DC"}],
"lastModifiedBy":"system",
"l2Interfaces": [{"_class":"L2Interface","guid":"FA048919AA953BA5A09580496017A776"},
{"_class":"L2Interface","guid":"297B125690B33B778C347E12CFC62689"}],
"createdBy":"system",
"_class":"LinuxUnitaryComputerSystem",
"controllers":[{"_class":"Controller","guid":"7B72D3B5448D30388F9D9497EA8F970D"},
{"_class":"Controller","guid":"B619ABB8B8343C1FAB5BF87AD425559E"}],
"guid":"C2D379A936433258BABBF682A8E71A82",
"CPUSpeed":3191000000,
"fqdn":"esx3-vm16-rhes4",
"contextIp":"9.43.73.87",
"OSInstalled":[{"_class":"Linux","guid":"04BFCBCD2A1733258F5C95CD281D91AF"}],
"memorySize":3988783104,
"ipInterfaces":[{"_class":"IpInterface","guid":"9CAA8E0197333BAD924EA3CCB1860920"},
{"_class":"IpInterface","guid":"C2E6D21CF24435EABCB8AA8136BB9F1B"}],
"signature":"9.43.73.87(000C29A467A9)",
"systemId":"2b095749",
"bidiFlag":3,
"name":"esx3-vm16-rhes4",
"OSRunning":{"_class":"Linux","guid":"04BFCBCD2A1733258F5C95CD281D91AF"},
"CPUType":"Intel(R) Xeon(TM) MV",
"type":"ComputerSystem",
"numCPUs":1,
"architecture":"i686",
"fileSystems":[{"_class":"UnixFileSystem","guid":"CDA94FB8C84B300ABA2A42E1EFEE6234"},
{"_class":"NFSFileSystem","guid":"6300742848BA39478EAEE4FB4709DF7A"}],
"lastModifiedTime":1225806427541
}]
```

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:

```
SELECT attribute-list FROM data-sources [ WHERE expression ]
```

Table 7 on page 36 describes the elements of an MQL query. MQL is not case-sensitive.

Table 7. MQL query elements	
Element	**Description**
attribute-list	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 the 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
data-sources	A comma-separated list of model object class names. These classes must be persistent, since you cannot query non-persistent objects.

expression	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 on page 36 for a description of operators and associated precedence.

Table 8 on page 36 describes the MQL operator precedence, with higher values representing greater precedence.

Table 8. MQL Operator Precedence		
Token	**Operator**	**Precedence**
or	logical OR	1
and	logical AND	2
instanceof	is instance of	3
==	equals	3

Table 8. MQL Operator Precedence (continued)		
Token	**Operator**	**Precedence**
!=	not equal to	3
>	greater than	3
>=	greater than or equal	3
<	less than	3
<=	less than or equal	3
starts-with	starts-with	4
ends-with	ends-with	4
equals	equals	4
not-equals	not equals	4
is-null	is null	4
is-not-null	is not null	4
in	in	5
()	parentheses	5
exists	array contains	5
upper()	function	5
lower()	function	5
!	unary not	5
.	dot selection	6
contains	contains	3
eval	eval	3

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:

```
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:

```
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.

```
statement := SELECT attribute-list [EXCLUDING attribute-list]
   FROM [ONLY] class_list { WHERE [expression |
   exists_expr] }
[ FETCH FIRST n { ROW | ROWS } [ ONLY ]] [ ORDER BY order_list ]
attribute_list := attrib {, attrib}* | *
class_list := domain_class {, domain_class }*
class := <a model object class>
exists_expr := exists( array_attrib op value {logical_op array_attrib op value}* )
expression := [ attrib op value | attrib post-op | pre-op ( attrib )
   |[ NOT ] IN ( expression [, ...] ){logical_op expression}*
value := <data value>
in_expr := [ NOT ] IN ( expression [, expression ... ])
array_attrib := <series of attributes where at least the second to last
   attribute is an array >
op := != | == | > | < | >= | <= | contains | starts-with | ends-with |
   equals | not-equals | instanceof | eval
logical_op := AND | OR | && | ||
post_op := is-null | is-not-null
pre_op := lower | upper
attrib := {class .} [<an attribute of a class>{.embedded_attribute} | * ]
embedded_attribute := [<embedded attribute>{.embedded_attribute} | *]
domain_class := {domain_list} class
domain_list := domain {, domain}* :
domain := <the server from which to pull data from, default: local database>
order_list := attrib [ ASC | ASCENDING | DESC | DESCENDING ] [ , order_list ]
```

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

Note: The `value` can be of the `attrib` type, when the basic operator op (!=|==|>|<|>=|<=) is used.

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 computers 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.

```
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:

```
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.

```
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:

```
./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:

```
./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:

```
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:

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

MQL queries with EVAL operator (XA and XD attributes only)

Many CDM attributes are moved into XML content of the XA or XD attributes. Therefore, MQL syntax supports a new operator `eval`, which can be added in the where clause. The `eval` operator enables querying CIs by the values of extended attributes or extended instances.

Note: All MQL queries examples contain escaped quotation marks (\"*value*\") because it is assumed that the queries are run in the following manner:

```
./api.sh -u username -p password find "MQL query"
```

For example, the following query was run to find a computer system with the `productID` attribute set to `'prod1'`:

```
SELECT * FROM ComputerSystem WHERE productID == 'prod1'
```

The following equivalent query uses the `eval` operator:

```
SELECT * FROM ComputerSystem WHERE XA eval '/xml
[attribute[@category=\"taddm_global\" and @name=\"productID\"]=\"prod1\"]'
```

The `eval` operator can be followed by any valid XPath expression that returns Boolean true or false value to enable the performance of correct SQL filtering by the persistence layer.

More examples

- Find all ComputerSystems, which have any extended attribute with the `val` value:
  - MQL:

    ```
    SELECT * FROM ComputerSystem WHERE XA eval '/xml[attribute=\"val\"]'
    ```

  - SQL:

    ```
    SELECT * FROM compsys WHERE xmlexists('$c/xml[attribute="val1"]'
    passing compsys.xa_x as "c")
    ```

- Find a ComputerSystem with the `attr2` extended attribute, which has the category set to `Other` and value set to `two`:
  - MQL:

    ```
    SELECT * FROM ComputerSystem WHERE XA eval '/xml/attribute
    [@name=\"attr2\" and @category=\"Other\" and text()=\"two\"]'
    ```

  - SQL:

    ```
    SELECT * FROM compsys WHERE xmlexists('$c/xml/attribute[@name="attr2"
     and and @category="Other" and text()="two"]' passing compsys.xa_x as
     "c")
    ```

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 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.ibm.cdb.service.registry.public.port` property.

   See the section on configuring the TADDM 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.

```
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:
 * <p>  get connection and log into api server
 * <p>  find all machines which have more than 1 CPU
 * <p>  find all Oracle instances
 */

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("\nall 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();
        }
      } catch (Exception ex) {
        System.err.println("exception:" + ex);
        ex.printStackTrace();
      }
    }

  }
}
```

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:

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

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

```
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:

```
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:

```
chmod +x build.sh
```

3) Run the build command:

```
./build.sh
```

b. On Windows:

1) Change to the %COLLATION_HOME%\sdk\examples\java directory.

2) Run the build command:

```
build.bat
```

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
```

Alternatively, run the following command:

```
run.sh FindXmlExample
```

These commands run the sample program and retrieve the XML data from the TADDM server.

Details about the sample Java application

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

```
/*
 * 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.ibm.cdb.service.registry.public.port in the $COLLATION_HOME/sdk/etc/collation.properties property file.
- The third and the fourth arguments control the SSL access.

```
/*
 * 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.

```
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 52

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 database to ensure that all changes 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:

  ```
  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:

  ```
  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:

```
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. If you receive an out of memory error message, increase memory available to Java virtual machine.

To increase the available memory, use the following values:

- On the TADDM server you can increase available memory by editing server's deployment configuration file::
  - Domain server: `cmdb-context.xml`
  - Synchronization server: `ecmdb-context.xml`
  - Storage Server: `storage-server-context.xml`
  - Discovery Server: `discovery-server-context.xml`

  In the appropriate file, locate the `jvmArgs` property for the JVM that is affected by out of memory error and increase the memory by changing the `DTaddm.xmx64` property.
- On the API client, increase the memory specified for the client application, for example in the `api.sh` or `api.bat` files.

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 on page 46 describes the change history methods you can use.

Table 9. Change history methods

Method	Description
`getChangeHistory(Guid[] Guids, long start, long end)`	`Return the change history for the specified array of GUIDs.`
`getChangeHistory(Guid[] Guids, long start, long end, int 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 Guid, long start, long end)`	Return the change history for the specified GUID.
`getChangeHistory(String[] classNames, long start, long end)`	Return the change history for the specified classes.
`getChangeHistoryByPersistTime(String[] classNames, long start, long end)`	Return the change history for the specified classes based on persist time.
`getChangeHistoryInXML(Guid[] Guids, long start, long end)`	Return the change history for the specified array of object IDs.
`getChangeHistoryInXML(Guid[] Guids, long start, long end, int 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 Guid, long start, long end)`	Return the change history for the specified GUID.

Table 9. Change history methods (continued)

Method	Description
`getPropagatedChanges(long primaryKey)` **Note:** This method is deprecated. Use the `getChangeHistory` method.	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(long primaryKey)` **Note:** This method is deprecated. Use the `getChangeHistory` method.	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.
`processChanges()`	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.
`getChangedClasses(long start, long end, int 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(long start, long end, int 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

Table 9. Change history methods (continued)	
Method	**Description**
`getChangeHistory(Guid[] Guids, long start, long end, int offset, int 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 on page 48 describes the discovery methods that you can use.

Many of the following methods refer to the load-balanced discovery. For more information, see "Load-balanced discover command" on page 104.

Table 10. Discovery methods	
Method	**Description**
`abortDiscovery()`	End the current discovery run.
`getStatus()`	Return the status of the current discovery run. The status can be any of the following values: • `Running` • `Idle`
`startDiscovery(String[] scope, String runName, String locationTag)`	Start a new discovery with a scope. The scope can be a name, or contain IP ranges, networks, and addresses, with specific IP addresses included and excluded.
`startDiscovery(RunDefinition runDef, String runName)`	Start a new discovery based on a discovery run definition that specifies profile, scope, run name and location tag information.
`startDiscovery(Guid[] guidList, String runName)`	Start a rediscovery of the objects with the specified globally unique identifiers (GUIDs).
`abortDiscovery(long runId)`	End the specified discovery run.
Fix Pack 2 `LoadbalancedDiscoveryStatus loadbalancedDiscoveryStatus()`	Return the status of the current load-balanced discovery run. The status can be any of the following values: • `Running` • `Idle`
Fix Pack 2 `startLoadbalancedDiscovery(String scopeGroupName, String discoveryPoolName, String profileName, String locationTag)`	Start a new load-balanced discovery against each scope included in the `scopeGroup` attribute, and on each server that belongs to the discovery server specified in the `poolName` attribute.

Table 10. Discovery methods (continued)	
Method	**Description**
Fix Pack 2 `startLoadbalancedDiscovery(String[] fileNames, int maxScopeSize, String profileName, String locationTag)`	Start a new load-balanced discovery against each scope starting from a specific file. Each file is parsed, and a new group named from file name (without extension) is created. In each group, scopes are added with the maximum size specified in the `maxScopeSize` attribute. The `poolName` attribute is assumed to be the same as the `groupName` attribute.
Fix Pack 2 `stopLoadbalancedDiscovery(String discoveryPoolName)`	Stop the load-balanced discovery for the specified `poolName` attribute.
Fix Pack 2 `pauseLoadbalancedDiscovery(String discoveryPoolName)`	Pause the load-balanced discovery for the specified `poolName` attribute. All scopes in the discovery run are moved back to the 'forTake' state. Current discoveries are aborted.
Fix Pack 2 `resumeLoadbalancedDiscovery(String discoveryPoolName)`	Return the LoadBalancedDiscoveryStatus object which exposes all discovery pools that are in progress, including pools that are in progress but paused. In addition, it returns information about which scopes are in progress, and which are waiting in the queue.
Fix Pack 3 `startDiscoveryRetID(String[] scope, String runName, String locationTag, String addressSpace)`	Start a new discovery with a scope and return the Discovery Run ID. The scope can be a name, or it can contain IP ranges, networks, and addresses with specific IP addresses included and excluded.

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 on page 49 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 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	
Method	**Description**
`find(Guid guid, int depth, Guid mss, String[] permissions)`	Return information about a specific managed element.
`find(Guid Guid, int depth, String[] permissions)`	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.

Table 11. Find methods (continued)

Method	Description
`find(String query, int depth, Guid mss, String[] permissions)` **Note:** This method is deprecated. Use `executeQuery` or a `find` method with a fillFlag parameter.	Return model objects matching the specified query. Note the following details about the depth parameter: • A depth of 0 returns an object with only its GUID set. • A depth of 1 returns all top-level attributes, along with contained objects having only their GUID attributes set. • Setting depth to DEPTH_INFINITE recursively locates all attributes until no additional objects are found. Cycles are avoided.
`find(String query, boolean fillFlag, Guid mss, String[] permissions)`	Return model objects matching the specified query. • query - Model language query to select and filter results. • fillFlag – Whether to populate all of the attributes. • mss - Managed Software System ID, or null if none • permissions - Optional list of permissions to restrict results
`findChanges(String query, int depth, long start, long end, int changeType)` **Note:** This method is deprecated. Use the `executeChangesQuery` method.	Return objects that have changed during the specified period for a given change type.
`findChanges(String query, boolean fillFlag, long start, long end, int changeType)` **Note:** This method is deprecated. Use the `executeChangesQuery` method.	Return objects that have changed during the specified period for a given change type.
`findChanges(String query, int depth, long start, long end, int changeType` **Note:** This method is deprecated. Use the `executeChangesQuery` method.	Return objects that have changed in the specified period for a given change type.
`findChanges(String query, boolean fillFlag, long start, long end, int changeType)` **Note:** This method is deprecated. Use the `executeChangesQuery` method.	Return objects that have changed in the specified period for a given change type.
`findCollections(Guid guid, String[] permissions)`	See the section on Managing collections.

Table 11. Find methods (continued)

Method	Description
`findImpactedBusinessApplications(Guid[] objects)`	See the section on Presentation.
`findImpactedBusinessServices(Guid[] objects)`	See the section on Presentation.
`findJDO(String root, String jdoQuery, String jdoVarDecl, int depth, Guid mss, String[] permissions)` **Note:** This method is deprecated. Use the `executeJDOQuery` method.	Same as `find(String, int, Guid)` except that the query must contain a Java Data Object query (JDOQL), with optional variable declarations.
`findJDO(String root, String jdoQuery, String jdoVarDecl, boolean fillFlag, Guid mss, String[] permissions)` **Note:** This method is deprecated. Use the `executeJDOQuery` method.	Same as `find(String, int, Guid)` except that the query must contain a Java Data Object query (JDOQL), with optional variable declarations.
`findRelationships(Guid managedElementGuid, int direction, String type, int scope, String[] permissions)`	See the section on Managing relationships.
`findRelationships(Guid[] sourceGuids, Guid[] targetGuids, String[] types, int comparisonFlags)`	See the section on Managing relationships.
`findXML(String query, int depth, int indent, Guid mss, String[] permissions)` **Note:** This method is deprecated. Use the `findXML(String,boolean,int,Guid,String[]` method.	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.
`findXML(Guid Guid, int indent, int depth, String[] permissions)`	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.
`findXML(String query, boolean fillFlag, int indent, Guid mss, String[] permissions)`	Returns an XML document containing a list of all objects which match the given query.
`findCount(String query, String[] permissions)`	Returns a count of the objects matched by the specified MQL query string.
`executeQuery(String query, Guid mss, String[] permissions)`	Executes an MQL query and returns a scrolling cursor interface.
`executeQuery(String query, int defaultDepth, Guid mss, String[] permissions)`	Executes an MQL query and returns a scrolling cursor interface.

Table 11. Find methods (continued)	
Method	**Description**
`executeChangesQuery(String query, int defaultDepth, long start, long end, int changeType, boolean deltaSynching)`	Returns objects that changed in the given period for a given change type. Otherwise, similar in functionality to `executeQuery (String,int,Guid,String[])`. If the **changeType** parameter is set to DELETED, then the WHERE clause in the query is ignored. All objects of the specified model object class that were deleted in the given time frame will be returned. The returned model objects are the shell model objects. They do not contain any attributes in them. Only the GUID is set. If the **changeType** parameter is set to ANYCHANGE, then in addition to executing the normal find query, the deleted shell objects will be returned. For the deleted objects, however, the where clause is ignored.
`findChangesForDeltaSynching(String query, boolean fillFlag, long start, long end, int changeType)` **Note:** This method is deprecated. Use the `executeChangesQuery` method.	Returns objects that were persisted in the data store during the specified period of time for the specified change type.

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 on page 52 describes the MSS methods you can use.

Table 12. Management Software System methods	
Method	**Description**
`deleteManagementSoftwareSystem(Guid guid)`	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.
`getManagementSoftwareSystemLinks(Guid guid, Guid mss, String[] permissions)`	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.

Table 12. Management Software System methods (continued)	
Method	**Description**
`getManagementSoftwareSystems(Guid guid, String[] permissions)`	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.
`registerManagementSoftwareSystem (ManagementSoftwareSystem 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 (ManagementSoftwareSystem 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. MSSObjectLink is not stored as a model object, therefore you are not able to use the `find` API to get the MSSObjectLink objects. Instead, use the APIs described in Table 13 on page 53, along with the getManagementSoftwareSystemLinks API.

Table 13. MSSObjectLink	
Method	**Description**
`getObjectSourceSystems(Guid[] objectId, String[] permission)`	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.

Table 13. MSSObjectLink (continued)	
Method	**Description**
`getObjectSourceSystems(Guid objectId, Guid mssGuid, String[] permission)`	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.
`getObjectSourceSystems(Guid mssGuid, String mssSourceToken)`	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).
`getObjectSourceSystems(String joinQuery)`	Returns an array of MSSObjectLinks that satisfy the JOIN query between MSSObjectLink and any other model objects. Since MSSObjectLinks will be stored directly as relational data rather than as a model object, you wont 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. **SELECT mssobjlink_rel.obj_x,** mssobjlink_rel.msssourcetoken_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. Vendor-supplied 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 on page 55 describes the access list methods you can use.

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

Table 14. Access list methods

Method	Description
`deleteDiscoveryAccessEntry(String authClassName, String name)`	Delete the access entry with the specified class and name.
`getAllDiscoveryAccessEntries()`	Get all the discovery access entries.
`getDiscoveryAccessEntry (String authClassName, String name)`	Get the discovery access entry with the specified class and name.
`updateDiscoveryAccessEntry(DiscoveryAccessEntry discoveryAccess)`	Update the properties of the access entry with the specified class and name.
`verifyDiscoveryAccessEntry(DiscoveryAccessEntry discoveryAccess)`	Verify whether the given properties match the current properties of the access entry with the specified class and name.

Example Java code

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

```
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) {
    ex.printStackTrace();
  } catch (Exception ex) {
    ex.printStackTrace();
  }
```

2. The following example Java code shows how to programmatically create discovery access entries:

```
try {
    // 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);

       // ccmsserver 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.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_ORDER, 2);
       entry.setProperty(DiscoveryAccessEntry.PROPERTY_USERNAME, "Administrator");
       entry.setProperty(DiscoveryAccessEntry.PROPERTY_PASSWORD, "password");
       entry.setProperty(DiscoveryAccessEntry.PROPERTY_AUTHTYPE,
      "authType_default");
       entry = api.updateDiscoveryAccessEntry(entry);
 } 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_CCMSSERVER.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();
       String name = (String) entry.getName();
    } catch (ApiException ae) {
       System.err.println("api exception:" + ae);
    } catch (Exception ex) {
       System.err.println("exception:" + ex);
    }
```

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

```
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:

```
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);

// This result should be false
boolean result = api.verifyDiscoveryAccessEntry(entry));

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

// This result should be true
result = api.verifyDiscoveryAccessEntry(entry));

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

// This result should be false
result = api.verifyDiscoveryAccessEntry(entry));

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

// This result should be true
```

```
   result = api.verifyDiscoveryAccessEntry(entry));
} 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:

```
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. Note that these methods are deprecated. For equivalent methods related to CustomCollections see the description of each deprecated method.

For related information, see “Managing grouping patterns” on page 74.

The Collection methods table describes the collection methods you can use.

Table 15. Collection methods

Method	Description
`addCollectionMembers(Guid collectionGuid, Guid[] guids)`	Add members to a collection. This method is unavailable in the Enterprise JavaBeans (EJB) interface. This method is deprecated. Use updateGroupingPattern(GroupingPattern groupingPattern) method followed by one of runPatternNow(...) methods instead.
`findCollections(Guid guid, String[] permissions)`	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. This method is deprecated. Use findCustomCollections(Guid guid, String[] permissions) method instead.
`removeCollectionMembers(Guid collectionGuid, Guid[] guids)`	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. This method is deprecated. Use updateGroupingPattern(GroupingPattern groupingPattern) method followed by one of runPatternNow(...) methods instead.

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 on page 59 describes the model methods you can use.

Table 16. Model management methods

Method	Description
`add(ModelObject[] obj, Guid mss)`	Add a new object to the database. **Note:** This method cannot be used to add GroupingPattern or Selector model objects. It throws ApiException when any of the provided objects is of the GroupingPattern or Selector type. Use GroupingPattern API for all operations on GroupingPatterns or selectors.
`addArrayElements(Guid object, String attrName, Guid[] elements, Guid mss)`	Add the elements to the named array of the specified object without fetching either the object or the array.
`compare(ModelObject left, ModelObject[] right, CompareOptions opts)` `compare(ObjectRef obj1, ObjectRef[] objs, CompareOptions opts)`	Compare a model object (or golden master) against a set of objects.

Table 16. Model management methods (continued)

Method	Description
`delete(Guid[] guids, Guid mss)` `delete(ModelObject[] obj, Guid 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: • 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. **Note:** This method cannot be used to delete GroupingPattern or Selector model objects. It throws ApiException when any of the provided objects is of the GroupingPattern or Selector type. Use GroupingPattern API for all operations on GroupingPatterns or selectors.

Table 16. Model management methods (continued)	
Method	**Description**
`deleteStale(Guid mss, long date)` **Note:** This method is deprecated.	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.
Fix Pack 2 `refresh(Guid mss, long date)`	Delete managed elements and relationships that have not been stored since a specified date. The managed elements and relationships that have an update time stamp less than 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 still 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.
`endBulkload(long bulkloadId)`	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.

Table 16. Model management methods (continued)	
Method	**Description**
`exportData(File directoryToWriteTo, long maxFileSize, Guid mss)`	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 maxFileSize 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.
`importData(URI source, boolean rebuildTopo, Guid mss)`	Convert XML data from the specified source into model objects which are updated within the TADDM database, according to the following rules: • When the source is a file, the contents of the single file are read and inserted. • When the source is a directory, each file in the directory is imported. • When the source is a remote object, such as an HTTP address, each update operates within its own transaction. Errors roll back the update of the current object only, and the import then proceeds.
`rebuildTopology()`	Rebuild the TADDM database derived data, such as dependencies, relationships and data consolidation. During this operation, the entire database is locked against updates.
`removeArrayElements(Guid object, String attrName, Guid[] elements, Guid mss)`	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.
`startBulkload(long timeoutInSeconds)`	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 `endBulkload()` method is called.

Table 16. Model management methods (continued)	
Method	**Description**
`update(ModelObject obj, mss)` `update(ModelObject[] obj, mss)`	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. 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. **Note:** This method cannot be used to update GroupingPattern or Selector model objects. It throws ApiException when any of the provided objects is of the GroupingPattern or Selector type. Use GroupingPattern API for all operations on GroupingPatterns or selectors.
`updateXML(String xml, Guid mss)`	Same as update(ModelObject[] obj, mss) except that the objects are represented as an XML string. **Note:** This method cannot be used to update GroupingPattern or Selector model objects. It throws ApiException when any of the provided objects is of the GroupingPattern or Selector type. Use GroupingPattern API for all operations on GroupingPatterns or selectors.

Example

This example illustrates how to compare objects.

```
//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 64 describes the relationship methods you can use.

Table 17. Relationship methods

Method	Description
`addRelationships(Relationship[] relationships, Guid mss)`	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).

Table 17. Relationship methods (continued)	
Method	**Description**
`deleteRelationships(Guid[] guids, Guid mss)` `deleteRelationships(String type, Guid source, Guid target, Guid 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.
`findRelationships(Guid managedElementGuid, int direction, String type, int scope, String[] 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

`findRelationships(Guid[] sourceGuids, Guid[] targetGuids, String[] types, int comparisonFlags)`
`findrelationships(managedElementGuid =>`
`findRelationships(managedElementGuid`
	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 on page 65 describes the session methods you can use.

Table 18. Session methods	
Method	**Description**
`close()`	Close a session.
`ApiFactory.getInstance().getApiConnection (String host, int port, String trustStoreLocation, boolean 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 (ApiConnection conn, String user, String password, long version)`	Return a Session object, which is used to execute TADDM API methods.

Table 18. Session methods (continued)	
Method	**Description**
`ApiFactory.getInstance().getSession (ApiConnection conn, long sessionId, long 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:

```
private ApiConnection conn_;
   try {
      conn_ = ApiFactory.getInstance().getConnection(
         "host.abcxyz.com",            //host name
              9433,                        //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:

```
private ApiConnection conn_;
   try {
      conn_ = ApiFactory.getInstance().getConnection(
         "host.abcxyz.com",            //host name
              9433,                        //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 port 9433 by default, though you can specify alternative port by specifying appropriate value 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 Portal, accessible at http://*server_name*:*web_server_port*, for example http://localhost: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;                  // 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 timout)
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 on page 67 describes the version methods you can use.

Table 19. Version methods

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

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 67 describes the metadata methods you can use.

Table 20. Metadata methods

Method	Description
`defineExtendedAttributeMeta(UserDataMeta udm)`	Adds or updates extended attributes meta.
`getAllMetaData()`	Returns metadata about Common Data Model without its Simplified Model part. It shows no classes of the Simplified Model and no moves or changes of attributes and relations.
`getAllMetaData(boolean flatten, Locale locale, boolean skipSimplifiedModel)`	Returns all metadata. This method is equivalent to find("ObjectClass", ...) where a cache of metadata is used on the server for faster access.
`getClassNames()`	Returns an array of model class short names, fully qualified name pairs.

Table 20. Metadata methods (continued)	
Method	**Description**
`getExtendedAttributeMeta(String classname)`	Retrieves extended attribute meta for a class. The method retrieves both `taddm_global` and custom category extended attributes meta. The `UserDataMeta` objects are collected from the specified class and all the elements that are higher in hierarchy than the specified class.
`getExtendedAttributes(Guid objGuid)`	Retrieves extended attribute values for an object. **Note:** This method is deprecated.
`getMetaData(String className)`	Returns the number, types, and names of all attributes in the model object. This method includes key/name rule, containment, relationship, and enumerated type information.
`removeExtendedAttributeMeta(String classname, Guid acct)`	Removes class-wide extended attributes or extended attributes for a specified class and account. **Note:** This method is deprecated.
`setExtendedAttributes(Guid objGuid, AttrNameValue[]`	Sets the values of the extended attributes. **Note:** This method is deprecated. Use the objects XA attribute instead.

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 on page 68 describes the presentation methods you can use.

Table 21. Presentation methods	
Method	**Description**
`findImpactedBusinessApplications(Guid[] objects)`	Determine the business applications that are affected based on the specified array of Configuration Items.
`findImpactedBusinessServices(Guid[] objects)`	Determine the business services that are affected based on the specified array of Configuration Items.
`getDetailsPanel(ObjectRef ref)`	Return the details panel for the specified object reference. An object reference is the combination of the object GUID and the version.
`getGraphView(ViewDefiner graphView)`	Return a graph for the specified ViewDefiner that describes the graph view.

Table 21. Presentation methods (continued)	
Method	**Description**
`getGraphViewImage(ViewDefiner graphView)` **Note:** Fix Pack 3 This method is deprecated.	Return an ImageStream object for the specified ViewDefiner that describes the graph view.
`getTreeView(ViewDefiner treeView)`	Return a TopologyTreeModel object for the specified ViewDefiner that describes the tree view.

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(
         new ObjectRef(guid,0));     //guid and version
      System.out.println("model is " + model);
   }
}
```

- The following example illustrates how to retrieve graph views.

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

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

- The following example 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 the access to one or more runtime operations.

Note: Security methods operate on CustomCollection objects with hierarchyType attribute set to "AccessCollection". They do not support old AccessCollection objects.

Table 22 on page 70 describes the security methods you can use.

Table 22. Security methods

Method	Description
`addAccess(Principal user, Resource resource, String role, String[] permissions)`	Add permission for a specific object to a role.
`addRuntimeAccess(Principal user, String role, String[] permissions)`	Add permission for one or more runtime operations to a role.
`assignPersonInRoleToAccessCollection (Person user, Role role, Guid[] guids, long[] versionId)`	Create an assignment (in potentially multiple versions) between a person in a role and a list of access collections.
`deleteAccess(Principal user, Resource resource, String role, String[] permissions)`	Delete a permission for a specific collection from a role.
`deleteRuntimeAccess(Principal user, String role, String [] permissions)`	Delete permission for one or more runtime operations from a role.
`getAccessDecisions(Principal user, Resource[] resources, String[] permissions)`	Determine whether the caller can access one or more objects with the specified permission.
`getDataPermissions(Principal user, Resource[] resources)`	Determine the data-level permissions that a user has for a set of objects.
`getEntitlements(Principal user, String[] permissions)`	Retrieve entitlements for the user. The entitlements are the objects that the user can access, based on the defined security policies.
`getRoles(Principal user)`	Retrieve the roles assigned to a user.
`getRuntimeAccess(Principal user)`	Retrieve permissions for runtime operations for a user.

Table 22. Security methods (continued)	
Method	**Description**
`getRuntimeAccessDecisions(Principal user, String[] permissions)`	Determine if a user has access to one or more runtime operations.
`removePersonInRoleToAccessCollection (Person user, Role role, Guid[] guids, long[] versionId)`	Remove an assignment in potentially multiple versions based on the specified person, role, and list of access collections to which the person-in-role is assigned.
`addAccess(Principal user, AccessDefinition[] accessDefinition)`	Add one or more objects (each with a set of permissions) to a role, as specified by each AccessDefinition object.
`addDataPermissionToRole(String role, String permission)`	Add a data permission to a role wherever the role exists in the stored policies.
`addRuntimePermissionToRole(String role, String permission)`	Add a runtime permission to a role.
`deletePermission(String permission)`	Remove a permission wherever the permission exists in the stored policies.
`deletePermissionFromRole(String role, String permission)`	Remove a permission from a role wherever the role exists in the stored policies.
`deleteRole(String role)`	Remove a role wherever the role exists in the stored policies.
`getEntitlementsForRole(Principal user, String role)`	Retrieve entitlements for the user in a specified role.

Managing application templates

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

Important: This API is deprecated for TADDM 7.3 and later. The GroupingPatternAPI can be used to manage Business Services. This API provides methods for creation, modification and deletion of Grouping Patterns. For more information, see "Managing grouping patterns" on page 74.

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 72 describes the application template methods that you can use.

Table 23. Application template methods

Method	Description
`createAppTemplate(String name, String type, boolean removeNonMembers, MQLRule[] operators)`	Creates a template with the specified rules. The following parameters are available: **name** The name of the application template to create. **type** 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. **removeNonMembers** 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. **MQLRule** An array that contains one or more rules.
`updateAppTemplate(String name, String type, boolean removeNonMembers, MQLRule[] operators)`	Updates a template with the specified rules. The following parameters are available: **name** The name of the application template to update. **type** 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. **removeNonMembers** 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. **MQLRule** An array that contains one or more rules.
`getAllAppTemplates()`	Retrieves all application templates.
`getAppTemplate(String name, int type)`	Retrieves the application template for specified name and type.
`removeAppTemplate(String name, int type)`	Removes the application template for specified name and type.
`removeMQLRule(String name)`	Removes an MQL rule. A rule can be removed only if it is not associated with any application templates.

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:

```
# 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
AA0A20EE5BBD336481279CA664FB380A:TADDM - Production
appTemplateName= str(appGuid) + ":" + BAname
appTemplate = api.createAppTemplate(appTemplateName, "APPLICATION", True,
jarray.array(rules,MQLRuleClass));
```

Example: Listing business applications

The following example lists business applications:

```
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.

```
# 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)
```

Managing grouping patterns

Grouping pattern methods enable creation, modification, deletion, and retrieval of grouping patterns together with their selectors.

For details, see the *Managing grouping patterns by using Java API* topic in the TADDM *User's Guide*.

Using the SOAP API

The Simple Object Access Protocol (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 75 describes the session requests you can use.

Table 24. Session requests		
Operation	**Input**	**Output**
`login` (Log in to the TADDM server)	`loginRequest` **user** The user name registered with the TADDM server **password** The password associated with the user **host** The name of the host, either as a name, or as an IP address (using dot notation) **port** The port number of the server	`loginReponse`
`logout` (Log out from the server)	`logoutRequest`	`logoutReponse`

Example

The following example shows a login XML request:

```
<?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 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 on page 76 describes the discovery requests you can use.

Table 25. Discovery requests		
Operation	**Input**	**Output**
`abortDiscovery` (Abort the currently running discovery)	`abortDiscoveryRequest`	`abortDiscoveryResponse`
`clearTopology` (Clear all discovery elements and relationships from the topology)	`clearTopologyRequest`	`clearTopologyResponse`
`getStatus` (Get the current discovery run status)	`getStatusRequest`	`getStatusResponse` `getStatusReturn`—String representation of current discovery run status
`rebuildTopology` (Rebuild the topology, including dependencies and relationships)	`rebuildTopologyRequest`	`rebuildTopologyResponse`
`startDiscovery` (Start a discovery using the specified scope)	`startDiscoveryRequest` **scope** The scope of the discovery: scope set or scope group name. **runName** The name of the discovery run.	`startDiscoveryResponse`
`startDiscoveryRetID` (Start a discovery using the specified scope)	`startDiscoveryRequest` **scope** The scope of the discovery: scope set or scope group name. **runName** The name of the discovery run.	`startDiscoveryResponse` run ID

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 77 describes the model and metadata requests you can use.

Table 26. Model and metadata requests

Operation	Input	Output
`exportData` (Export all top level objects in the TADDM database to a specified directory in XML format)	`exportDataRequest` **directoryToWriteTo** The name of the directory to which the data is to be exported **maxfilesize** The maximum file size to export **mssGuid** The GUID of the Management Software System (MSS)	`exportDataResponse`
`exportDataUsingMssName` (Export all top-level objects in the TADDM database to a directory, in XML format, specifying the MSS using a name)	`exportDataUsingMss NameRequest` **directoryToWriteTo** The name of the directory to which the data is to be exported **maxfilesize** The maximum file size to export **mssGuid** The GUID of the Management Software System (MSS)	`exportDataUsingMss NameResponse`
`getClassNames` (Get all class names in the model that can be used in the query language)	`getClassNamesRequest`	`getClassNamesResponse` `getClassNamesReturn`: Array of model class short name/fullly qualified name pairs
`importData` (Convert XML data from the specified source into model objects and update the TADDM database)	`importDataRequest` **source** The source from which to import the data **rebuildTopo** A Boolean to rebuild the topology **mssGuid** The GUID of the Management Software System (MSS)	`importDataResponse`

Table 26. Model and metadata requests (continued)		
Operation	**Input**	**Output**
`importDataUsingMssName` (Convert XML data from the specified source into model objects and update the TADDM database, specifying Management Software System by name)	`importDataUsingMss NameRequest` **source** The source from which to import the data **rebuildTopo** A Boolean to rebuild the topology **mssName** The name of the Management Software System (MSS)	`importDataUsingMss NameResponse`
`insert` (Insert or update the model object, specified in XML format, in the TADDM database)	`insertRequest` **xml** The model object in XML format **mssGuid** The GUID of the Management Software System (MSS)	`insertResponse`
`insertUsingMssName` (Insert the model object, specified in XML format, specifying the Management Software System by name)	`insertUsingMss NameRequest` **xml** The model object in XML format **mssName** The name of the Management Software System (MSS)	`insertUsingMss NameResponse`

Example

The following example shows a getClassNames XML request:

```
<?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: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 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:getClassNamesResponse soapenv:encodingStyle=
         "http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://localhost">
      <getClassNamesReturn xsi:type="soapenc:string" xmlns:soapenc=
         "http://schemas.xmlsoap.org/soap/encoding/">
      AbstractResource,
      com.collation.platform.model.topology.process.AbstractResource,
```

```
      Accepts,
      com.collation.platform.model.topology.relation.Accepts,
       .
       .
       .
      </ns1:getClassNamesResponse>
   </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 on page 79 describes the find operations you can perform.

Table 27. Find requests

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

Table 27. Find requests (continued)		
Operation	**Input**	**Output**
`findUsingGuid` (Find using the GUID of the model object)	`findUsingGuidRequest` **guid** The GUID against which to execute the find **depth** The level of the result tree to construct **indent** The indentation to use for the resulting XML file	`findUsingGuidResponse` `findUsingGuidReturn`: XML representation of the query results
`findUsingMssName` (Find using the Model Query Language, specifying the Management Software System by name)	`findUsingMssNameRequest` **query** The query string **depth** The level of the result tree to construct **indent** The indentation to use for the resulting XML file **mssName** The name of the Management Software System (MSS)	`findUsingMssNameResponse` `findUsingMssNameReturn`: XML representation of the query results

Example

The following example shows a find XML request:

```
<?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>
    <multiRef id="id1" soapenc:root="0" soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/" xsi:type="soapenc:int"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">4</multiRef>
    <multiRef id="id0" soapenc:root="0" soapenv:encodingStyle=
       "http://schemas.xmlsoap.org/soap/encoding/" xsi:type="soapenc:int"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">2</multiRef>
   </soapenv:Body>
</soapenv:Envelope>
```

The following example shows the find XML response:

```
<?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:findResponse soapenv:encodingStyle=
         "http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns1="http://localhost">
```

```
         <findReturn xsi:type="soapenc:string" xmlns:soapenc=
            "http://schemas.xmlsoap.org/soap/encoding/">
         &lt;?xml version=&quot;1.0&quot; encoding=&quot;ISO-8859-1&quot;?&gt;
         &lt;results
         xmlns=&quot;urn:www-collation-com:1.0&quot;
          .
          .
          .
         &lt;architecture&gt;Intel&lt;/architecture&gt;
         &lt;/ComputerSystem&gt;
         &lt;/results&gt;
         </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 on page 81 describes the change history requests you can use.

Table 28. Change history requests

Operation	Input	Output
`getChangeHistory` (Get the change history for the start and end period using the specified GUID)	`getChangeHistoryRequest` **guid** The GUID of the object for which the change history is required **start** The start time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT **end** The end time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT	`getChangeHistoryResponse` `getChangeHistoryReturn`: XML representation of a list of ChangeHistory objects
`getChangeHistory` (Get the change history for the start and end period for multiple GUIDs)	`getChangeHistoryRequest1` **guids** The list of comma-separated GUIDs of the objects for which the change history is required **start** The start time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT **end** The end time for the change period, specified in milliseconds since January 1, 1970, 00:00:00 GMT	`getChangeHistoryResponse1` `getChangeHistoryReturn`: XML representation of a list of ChangeHistory objects

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 on page 82 describes the version requests you can use.

Table 29. Version requests

Operation	Input	Output
`createVersion` (Create a named snapshot of the current TADDM database data)	`createVersionRequest` **name** The name of the version **description** A description of the new version	`createVersionResponse`
`createEmptyVersion` (Create an empty version, with no data, in the TADDM database)	`createEmptyVersionRequest` **name** The name of the version **description** A description of the new version	`createEmptyVersionResponse`
`deleteVersion` (Delete a version from the TADDM database)	`deleteVersionRequest` **versionID** The identifier of the version	`deleteVersionResponse`
`deleteVersionUsingName` (Delete a version, identified by name, from the TADDM database)	`deleteVersionUsingNameRequest` **versionName** The name of the version	`deleteVersionUsingNameResponse`
`getAllVersions` (Get the names of all defined TADDM database data versions)	`getAllVersionsRequest`	`getAllVersionsResponse`

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 `$COLLATION_HOME/deploy-tomcat` directory (TADDM 7.3.0) or `$COLLATION_HOME/apps` (TADDM 7.3.0.1, and later) 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

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&fetc
hSize=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 `$COLLATION_HOME/etc` directory on the TADDM server.

Procedure

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:

```
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);
        URL url = new
          URL("https://cab.tivlab.austin.ibm.com:9431/rest/model/"+
              "Repository?depth=1&feed=json");
        HttpsURLConnection urlConn = (HttpsURLConnection) url.openConnection();

        System.out.println("sending request...");
        urlConn.setRequestMethod("GET");
        urlConn.setAllowUserInteraction(false); // no user interaction
        urlConn.setDoOutput(true); // want to send
        urlConn.setRequestProperty( "Content-type", "text/xml" );
        urlConn.setRequestProperty( "accept", "text/xml" );
        urlConn.setRequestProperty( "authorization", "Basic " +
            encode("administrator:collation"));
        Map headerFields = urlConn.getHeaderFields();
        System.out.println("header fields are: " + headerFields);

        int rspCode = urlConn.getResponseCode();
        if (rspCode == 200) {
            InputStream ist = urlConn.getInputStream();
            InputStreamReader isr = new InputStreamReader(ist);
            BufferedReader br = new BufferedReader(isr);

            String nextLine = br.readLine();
            while (nextLine != null) {
                System.out.println(nextLine);
                nextLine = br.readLine();
            }
        }
```

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 in the IBM SDK Java Technology Edition version 5 and later. JXPath support is included with the TADDM SDK.

Procedure

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:

  – `$COLLATION_HOME/log/tomcat.log` (TADDM 7.3.0)
  – `$COLLATION_HOME/log/wlp.log` (TADDM 7.3.0.1, and later)
  – `$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`:

  ```
  http://example.com:9430/rest/model/ComputerSystem?depth=2&feed=xml&OSRu
  nning.OSName=Linux&position=5
  ```
- 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=xm
  l&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:

```
[{"signature":"JsonRestExample1","_class":"LinuxUnitaryComputerSystem","
numCPUs":2,"guid":"cs1","OSRunning":{"_class":"Linux","parent":"cs1","na
me":"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/1D646C44FDEB3857B40B98BD
  F9C0F407?mssGuid=CF5EBF574E7F382289B3F35FB5776628
  ```

- 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
  ```

Maintaining grouping patterns using REST API
Grouping patterns can be maintained without the need to use UI but by using REST API.

REST API for grouping patterns can be accessed by using standard login credentials. Basic WWW authentication is needed. If you want to access REST API through a web browser with existing user session, additional logon is not needed.

Data can be sent and received in two formats: JSON and XML (application/json and application/xml). The path to the service is `/cdm/api/groupingpatterns`.

The following list consists of methods that you can use and the actions that they perform:

- `/cdm/api/groupingpatterns :: method GET` - returns all grouping patterns.
- `/cdm/api/groupingpatterns/{GUID} :: method GET` - returns grouping patterns with the given GUID.
- `/cdm/api/groupingpatterns :: method POST :: load in JSON or XML format` - creates new grouping patterns.
- `/cdm/api/groupingpatterns/{GUID} :: method DELETE` - deletes grouping patterns with the given GUID.
- `/cdm/api/groupingpatterns/ :: method PUT :: load in JSON or XML format` - updates grouping patterns.

Example load in XML format:

```
<groupingPattern name="GroupingPattern1" hierarchyType="ACCESS_COLLECTION"
active="true">
    <selector name="Selector1" lowerDown="include" lowerUp="include"
higherDown="include"
 higherUp="include" GroupingNameExpression="GroupingNameExpression"
useTraversalTemplate="false" type="MQL">
        <query>ComputerSystem where guid == '00000000000000000000000000000000'</
query>
    </selector>
    <description>GroupingPattern1Description</description>
</groupingPattern>
```

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 sending POST request, specifying a name for the discovery run.

  You can use this resource to start discovery with or without a profile.

  For example, this HTML form starts discovery with the 'TestRun2' name by using the specified profile for specified scope.

  ```
  <form action="http://example.com:9430/rest/discovery/start/TestRun2"
  method="post">
       Profile: <input type="text" name="profile"><br>
       Scope: <input type="text" name="scope"><br>
       <input type="submit" value="Submit"></input>
  </form>
  ```

- 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|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.

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&OSRu
nning.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`.

```
http://example.com:9430/rest/model/MQLQuery?query=select%20displayName,
OSRunning.OSName%20from%20ComputerSystem&position=2&fetchSize=2&feed=xm
l&depth=2&position=4
```

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/ModelObject/1D646C44FDEB3857B40B98BD
F9C0F407?mssGuid=CF5EBF574E7F382289B3F35FB5776628
```

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:

```
[{"type":"java.lang.String","column":"BOOTORDER_X","length":192,"name":
"bootOrder","arrayType":false,"_class":"ObjectAttribute","timestampType
":false,"displayString":"Boot Order"}, {"type":"com.collation.platform.
model.topology.sys.zOS.ZReportFile","table":"COMPUTERSYSTILES_935A6002X
","column":"PK__ZREPORTFILES_X","length":192,"name":"ZReportfiles","arr
ayType":true,"reverseRelationship":true,"_class":"ObjectAttribute","rel
ationshipType":"com.collation.platform.model.topology.relation.AppliesT
o","timestampType":false,"displayString":"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.Runs
On</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:

```
[{"description":"Validated on February 4","_class":"Linux","guid":"347EE
64E4FA93139A581757EC7F3ED2D"}]
```

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.

locationTag=*value*
The value of location tag to set for objects that are created during a discovery.

Fix Pack 3 **addressSpace=*value***
The address space name that is set for all IpAddress or IpNetwork objects that are created during a discovery.

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:

```
[{“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:

```
[{“name”:”scope1”},{“name”:”scope2”}]
```

Example

This example lists discovery scopes:

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

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 set or scope group.

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 set or scope group.

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*`] [-T|--truststorefile]` *COMMAND COMMAND-PARAMETERS*

For Windows:

`api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile]` *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`. If you use the **-T** parameter, you must also specify the **-H** parameter.

`-P|--port` *port*
Optional: The TADDM server port, by default, is 9433.

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

`-T|--truststorefile` *truststore*
Optional: Location of the truststore file, `jssecacerts.cert`, with a certificate for connection to the TADDM server. This parameter is required for secure connection to TADDM. If you use this parameter, you must also specify the **-H** parameter.

`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*`] [-T|--truststorefile] 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.

Note: If you want to run an advanced change history query on the discovery server, the `-H` parameter must point to a storage server. Otherwise, the query fails.

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
 10A5794E86C53A0BBB10F262055CB3EA “06/06/05 12:00:00 AM” “06/08/05 12:00:00 AM”
```

Delete command
The **delete** command removes the objects from the TADDM database.

Command syntax

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] delete` *guid1* `[`*guid2 guid3 ... guidn*`]`

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] delete -f|--file` *guid-list-file*

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] delete -m|--mql` *mql-query*

Parameters

delete
Runs the **delete** command.

guid1 [guid2 guid3 ... guidn]
Are the GUIDs of the objects to delete.

-f|--file ***guid-list-file***
Is the location and name of the text file that contains the GUID list of the objects to delete. The GUIDs can be separated by space, tab, or the new line character. The file can be generated by the `dbquery` script.

-m|--mql ***mql-query***
Is the MQL query, which selects the objects to delete.

Example

The following 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
 10A5794E86C53A0BBB10F262055CB3EA
```

The following command deletes objects with specified GUIDs. Enter the command on one line.

```
api.sh -u user -p password -H host -P port delete
 C172810FD1CF3E108B8127BC47D2667B 059E4D85B34C32D1B5A80D9E2DB09EBD
```

```
35D21D3CA08539908DC1762D26897FB6
```

The following command deletes objects that are selected by the specified MQL query. Enter the command on one line.

```
api.sh -u user -p password -H host -P port delete
 --mql "select * from AppServer where objectType == 'SAS'"
```

The following command deletes objects based on the GUID list in the text file. The dbquery script is used to generate guid-list-file. Enter the command on one line.

```
dbquery.sh -u user -p password -q "select guid_c from BB_APPSERVER6_V
where objectType_C like '%SAS'" > /tmp/guidListToDelete.txt
api.sh -u user -p password -H host -P port delete -f /tmp/guidListToDelete.txt
```

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*] [-T|--truststorefile] discover start [--name *run-name*] [--profile *profile-name*] [--locationTag *location-tag*] [-a|-addressSpace *addressSpace*] *scope-element1*|*scope-set1*|*scope-group1 scope-element2*|*scope-set2*|*scope-group2 ... scope-elementn*|*scope-setn*|*scope-groupn*

api.sh|api.bat -u|--user *user* -p|--password *password* [-H|--host *host*] [-P|--port *port*] [-T|--truststorefile] discover abort|status

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)

start ***scope-set1 scope-set2 ... scope-setn***
Starts a discovery with the specified scope sets.

start ***scope-group1 scope-group2 ... scope-groupn***
Starts a discovery with the specified scope group.

--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.

--locationTag ***location-tag***
Specifies location tag that is used for this discovery.

Fix Pack 3 **-a | -addressSpace** ***addressSpace***
Specifies the address space name for all IpAddress or IpNetwork objects that are created during the discovery started with **api.sh** or **api.bat**.

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

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 set 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 set 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"
```

- This command discovers the scope set named MyScopeSet using a Level 1 discovery profile. Enter the command on one line:

```
api.sh -u user -p password discover start
 --profile “Level 1 Discovery” "MyScopeSet"
```

- This command discovers the scope group named MyScopeGroup using a Level 1 discovery profile. Enter the command on one line:

```
api.sh -u user -p password discover start
 --profile “Level 1 Discovery” "MyScopeGroup"
```

Fix Pack 2 **Load-balanced discover command**

The **discoverloadbalanced** command allows a discovery to be run in load-balancing mode, which means discovery is continuous.

About load-balanced discovery

Load-balanced, continuous discovery is achieved by using a pool of discovery servers. This maximizes the utilization of servers and guards against failover, thereby preventing discoveries in progress from being interrupted.

Discovered environments can be separated into areas, with each drawing on its own pool of servers. Furthermore, each discovery can be performed by entering sets of IP addresses to define the scope of the discovery.

The PrimaryStorageServer (PSS) acts as a load-balancing discovery controller, allocating scope and resources to each discovery server (DS). Should the PSS fail, all running discoveries will be completed, but no new ones allocated.

The PSS controls discovery by placing work into a queue, but does not push tasks to a DS; instead, each participating DS actively requests work from the queue when it is ready, and then signals 'in progress' as the discovery progresses. If no 'in progress' signal is received, the PSS reallocates the work.

When running in continuous (or load-balanced) mode, the discovery server starts another discovery as soon as DiscoveryWorker threads are free to be used, even before the previous discovery is finished. The

continuous discovery command is available only to the primary storage server (in enterprise mode) and domain server (in single domain mode).

Tip: Sensors storing data are displayed on the UI as 'in progress'. This is **not** the same as the number of threads running a discovery.

Load-balancing discovery scenarios

Load-balanced HA discovery based against scope group
A user runs a discovery against a defined scope group via command line interface (CLI). The pool of servers is used to optimize the discovery workload, based on the scopes (from single to multiple scope) being part of the scope group.

Dynamic generation of the scope group

A user runs a load-balanced discovery using a list of IP addresses provided as a file. This creates a scope group with automatically generated child scopes, with IP addresses divided between child scopes.

Command syntax

As for a standard discovery using the **`discover`** command, the continuous load-balanced discovery functionality (**`discoverloadbalanced`**) is controlled using **`api.sh`**.

Parameters

api.sh -u <user> -p <password> discoverloadbalanced start --poolName <poolName> --scopeGroup <scopeGroup> [--profile <profileName>]
Starts continuous, load-balanced discovery against each scope in `<scopeGroup>`, and on each server belonging to the discovery pool specified in `<poolName>`

api.sh -u <user> -p <password> discoverloadbalanced startFromFiles --files <filePath1,filePath2,...> --maxScopeSize <size> [--profile <profileName>]
Starts continuous, load-balanced discovery against each scope passed via `--files` argument. Each file is parsed, and a new group named from file name (without extension) is created. In each group scopes are added with the maximum size specified in `maxScopeSize`. `poolName` is assumed to be the same as `groupName`

api.sh -u <user> -p <password> discoverloadbalanced startFromDirectory --dir <directory path> --maxScopeSize <size> [--profile <profileName>]
As in the `--files` example, but starts discovery against each file in the directory.

api.sh -u <user> -p <password> discoverloadbalanced status
Prints the current status of the load-balanced discovery.

api.sh -u <user> -p <password> discoverloadbalanced abort <poolName>
Aborts the discovery for the `poolName` specified.

api.sh -u <user> -p <password> discoverloadbalanced abort --poolName <poolName> --scopeSet <scopeSetName>
Aborts the discovery for a specified `scopeSet` running or scheduled within the specified `poolName`.

api.sh -u <user> -p <password> discoverloadbalanced pause <poolName>
Stops discovery for the specified `poolName`, and all scopes in the discovery run are moved back to the 'forTake' state. Current discoveries are aborted.

api.sh -u <user> -p <password> discoverloadbalanced resume <poolName>
Resumes paused discovery for the specified `poolName`, and all scopes in the 'forTake' state are now available for processing.

Properties for the primary storage server

Set the following properties for the primary storage server (or domain):

com.ibm.cdb.internalscheduling.discoverypool.scopePrefix
Specifies prefix for automatically created scopes (for startFromFiles and startFromDirectory options)

Default: `com.ibm.cdb.internalscheduling.discoverypool.scopePrefix=_auto_`

com.ibm.cdb.internalscheduling.discoverypool.timeToNonAlive
Specifies time in seconds, that is, how long discovery is assumed to be running without any information about status received from the discovery server.

Once time has passed, the scope is moved back to the 'forTake' state, so it can be redone by the other server which is still responding.

Default: `com.ibm.cdb.internalscheduling.discoverypool.timeToNonAlive=180`

Properties for the discovery server

Set the following properties for the discovery server (or domain):

com.ibm.cdb.internalscheduling.discoverypool.enabled
Enables the discovery server to be part of the discovery pool.

Default: `com.ibm.cdb.internalscheduling.discoverypool.enabled=false`

com.ibm.cdb.internalscheduling.discoverypool.name
Defines poolName for a discovery server

Default: `com.ibm.cdb.internalscheduling.discoverypool.name=DEFAULT`

com.ibm.cdb.internalscheduling.discoverypool.checkinginterval
Specifies the time in seconds that the discovery server checks for new jobs to take (if ready), also how often to inform PSS about a job in progress.

The interval specified needs to be smaller than `com.ibm.cdb.internalscheduling.discoverypool.timeToNonAlive` on a primary storage server, otherwise scopes might be returned to be re-executed.

Default: `com.ibm.cdb.internalscheduling.discoverypool.checkinginterval=20`

Note: Logs from the continuous discovery are stored in the ApiServer.log on the domain, or the primary storage server.

Export command

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

Command syntax

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] export [--mssguid` *mss-guid*`|--mssname` *mss-name*`] [--maxfilesize` *size*`]` *local-directory-to-write-data*

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.

You can find the Management Software System name in the UI. Go to the **Details** pane of a model object and open the **MSS Info** tab. **Subcomponent Instance Name** is the Management Software System name, for example `LinuxComputerSystemSensor`. If you want to export all objects that are discovered by this sensor, run the following command:

```
export --mssName LinuxComputerSystemSensor
```

If you want to find both name and GUID of the Management Software System, run the following command:

```
api.sh -u user -p password find -d 1 ManagementSoftwareSystem
```

As a result, you get a list of model objects with a specific set of information, for example:

```
<ManagementSoftwareSystem array="1"
    guid="MY_GUID" xsi:type="coll:com.collation.platform.model.topology.process.
ManagementSoftwareSystem">
    <manufacturerName>IBM</manufacturerName>
    <productName>TADDM</productName>
    <hostname>hostname.domain</hostname>
    <subcomponent>Discovery</subcomponent>
    <subcomponentInstanceName>IvmSensor</subcomponentInstanceName>
    <displayName>IBM:TADDM:hostname.domain:Discovery:IvmSensor</displayName>
    <bidiFlag>3</bidiFlag>
</ManagementSoftwareSystem>
```

In this example, the MSS GUID is `MY_GUID` and the MSS name is `IvmSensor`. If you want to run the **`export`** command with this data, use one of the following commands:

```
export --mssguid MY_GUID
```

or

```
export --mssname IvmSensor
```

Note: In case of GUID, use the correct value from the API query.

`--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*`] [-T|--truststorefile] find [--depth` *depth*`] [--indent` *num-spaces*`] [-o|--outfile` *local-file-to-write-to* `[-x --maxfilesize` *size*`]] [-s --suppress` *list-of-classes-to-suppress*`] root`

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] find [--depth` *depth*`] [--indent` *num-spaces*`] --guid` *object-guid*

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

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] find [--depth` *depth*`] [--indent` *num-spaces*`] [--mssguid` *mss-guid*`|--mssname` *mss-name*`] mql-query`

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] find --count` *mql-query*

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. If possible, do not use depth value higher than 3. You can increase the memory with the `-Xmx` 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

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] import [--timeout` *time*`] [--mssguid` *mss-guid* `--mssname` *mss-name*`] [--maxfilesize` *size*`]` *local-directory-to-read-data-from*

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/
```

Merge command

The **merge** command merges CI on the basis on their GUIDs.

Command syntax

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] merge` *DurableCI_GUID TransientCI_GUID* `[type` *type*`]`

Parameters

merge
Runs the **merge** command.

DurableCI_GUID
The GUID that persists after merging with the other one.

TransientCI_GUID
The GUID that is merged into the durable GUID and removed from the database.

type ***type***
The type of the merge process. There are two possible values:

- *0*, which stands for a shallow type of the merge process, meaning that only the main objects are merged. The child objects of the transient GUID are replaced with the child objects of the durable GUID. The shallow type of the merge process is the default type.
- *1*, which stands for a deep type of the merge process, meaning that main objects along with their child objects are merged.

Note: At this moment, only shallow type of the merge process is enabled. When you specify *1* for the **type** parameter, still the shallow type is used. The deep type will be enabled in future.

Important: When you merge CIs, both of the CIs must be of the same class. For example, you can merge two ComputerSystem objects, but you cannot merge a ComputerSystem object with an ApplicationServer object.

Example

The following command merges two specified GUIDs by using the shallow type of the merge process. Enter the command on one line.

```
api.sh -u user -p password -H host -P port merge 10A5794E86C53A0BBB10F262055CB3EA
C172810FD1CF3E108B8127BC47D2667B type 0
```

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] [-P|--port
port] [-T|--truststorefile] naming -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
```

Rediscover command

The **rediscover** command rediscovers objects.

Note: Before you run the rediscovery, make sure that the **com.collation.rediscoveryEnabled** parameter is set to `true`. The objects that you want to rediscover must be previously discovered at least once.

Command syntax

```
api.sh|api.bat -u|--user user -p|--password password rediscover guid1 [guid2 guid3 ...
guidn]
```

Parameters

rediscover
Runs the **rediscover** command.

guid1 [guid2 guid3 ... guidn]
Are the GUIDs of the objects to rediscover.

Example

The following command rediscovers objects with specified GUIDs. Enter the command on one line.

```
./api.sh -u administrator -p collation rediscover 4C778F231DB03FCF815E38EAD7CB1D66
B609D10039C23AE9A51E433EC311A9EE F503F3B70DF93587A635D574A78B248A
```

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*`] [-T|--truststorefile] servers getservers`

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

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

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

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

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

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

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

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

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

`api.sh|api.bat -u|--user` *user* `-p|--password` *password* `[-H|--host` *host*`] [-P|--port` *port*`] [-T|--truststorefile] 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 the synchronization 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 shows the status of topology groups.

Command syntax

```
api.sh|api.bat -u|--user user -p|--password password [-H|--host host] [-P|--port
port] [-T|--truststorefile] 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 TADDM.

Command syntax

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

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

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

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

Limitation: You cannot use script-based sensors to create custom server extensions.

Limitation: When you extend the discovery on the Windows operating systems to run commands which return the output with Unicode characters, such characters are not stored.
Fix Pack 3 In TADDM 7.3.0.3, and later, storing the Unicode characters is supported. However, you must first send the output to a file and then read the file. The following example shows commands, which you can use to perform these two operations. In the example, the output of the **unicodetest.bat** command is sent to the `c:\\r.txt` file.

```
os_handle.executeCommand("c:\\unicodetest.bat | out-file c:\\r.txt")
output =  os_handle.executeCommand("cmd.exe /u /c type c:\\r.txt")
```

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” on page 115.
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 115. To see a sample application, refer to “Sample custom server extension application” on page 140.
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 115 describes the settings that you can specify for extended attributes.

Table 30. Extended attributes

Field	Description
Component type	The type of component.
Extended attribute name	The name of the extended attribute.
Extended attribute type	The type for the extended attribute.
Inherited attribute name	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.
Inherited attribute type	The type of the inherited attribute.

Procedure

To specify extended attributes, complete the following steps:

1. Launch the Data 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 Jython 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.

You can also extend custom server and computer system templates by running Jython scripts. For details, see the *Extending custom server and computer system templates* topic in the TADDM *User's Guide*.

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 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 on page 117 describes the functions you can use.

Table 31. Capability functions

Function	Description
`getSimpleCapabilitiesFactory`	Returns SimpleCapabilitiesFactory for a given IP address.

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 on page 117 describes the functions you can use.

Table 32. Command and process functions

Function	Description
`addProcessToPool`	Add a runtime process to a process pool.
`executeCommand`	Run a command on the target.
`executeCommandWithTimeout`	Run a command on the target with a timeout that specifies how long the command is permitted to run.
`getPidConnectionMap`	Return a Python dictionary of Python lists containing the following information: • keys: process IDs • lists: TCP connections of the process IDs
`getPidPortList`	Return a Python dictionary of Python lists containing the following information: • keys: process IDs • lists: the ports the process is using either for listening or connecting
`getPidToRuntimeProcessMap`	Return a Python dictionary containing the following information: • process IDs • runtime process information
`getProcessByPid`	Return the CDM RuntimeProcess object associated with a given process ID.

Table 32. Command and process functions (continued)	
Function	**Description**
`getServerProcesses`	Return a Python dictionary of Python lists containing the following information: • keys: process IDs • lists: bind addresses of the listen ports

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 on page 118 describes functions you can use.

Table 33. Common Data Model functions	
Function	**Description**
`cloneModelObject`	Create a copy of a CDM ModelObject.
`newModelObject`	Create a CDM object.
`setExtendedAttributes`	Set the values of the extended attributes.

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 on page 118 describes functions you can use.

Table 34. DNS functions	
Function	**Description**
`getLocalDNSLookup`	Perform a name lookup on the TADDM server.
`getRemoteDNSLookup`	Perform a lookup of a name extracted from a remote configuration which may not resolve on the TADDM server.
`validateFqdn`	Check an FQDN to ensure it conforms to the rules outlined in RFC 1035.

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 119 describes the functions you can use.

Table 35. File access functions	
Function	**Description**
`getFile`	Capture a file from a remote target and return the file contents and metadata.
`getFileWithLengthLimit`	Capture a file, up to the specified maximum length, from a remote target and return the file contents and metadata.
`listDirectory`	Return a Python list containing the contents of a directory on a remote target.

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 on page 119 describes the functions you can use.

Table 36. IP and MAC address functions	
Function	**Description**
`binToDot`	Convert a binary representation of an IP address to dot notation.
`bitsMaskToDottedDecimalMask`	Convert network bits mask notation to dotted decimal notation, for example 24 to 255.255.255.0.
`calcNetworkAddress`	Calculate the network address given an IP address and netmask.
`canonicalMac`	Remove separators or radix notation from a MAC address and return the hexadecimal MAC address as a string with alpha characters capitalized.
`classlessNotation`	Calculate the classless notation of an IP network.
`dotToBin`	Convert an IPv4 address from dot notation to binary form.
`ipInSubnet`	Determine if an IP address is a member of a given subnet and not the broadcast address.
`networkToList`	Return all IP addresses that are members of the CDM representation of the IpNetwork parameter.
`validateIp`	Validate an IP address in dot notation.

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 120 describes the functions you can use.

Table 37. Operating system functions	
Function	**Description**
`getAppTarget`	Return 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
`getCsTarget`	Return 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
`getComputerSystem`	Return an object to which the OS handle is connected with the attributes populated.
`getNewOsHandle`	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.
`getOperatingSystem`	Return an object representing the operating system to which the OS handle is connected.
`queryRegistry`	Return a registry key as XML.

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 on page 120 describes the functions you can use.

Table 38. Path functions	
Function	**Description**
`unixSlashes`	Substitute Windows path separator characters for Unix path separators
`windowsSlashes`	Substitute Unix path separator characters for Windows path separators

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. You can create 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 121 describes the functions you can use.

Table 39. Utility functions

Function	Description
`findElementsForXPath`	Queries objects and returns a collection of objects from the query.
`getArray`	Get an array in Python for use with certain Java and TADDM functions.
`init`	Initialize the custom server extension API.
`splitArgs`	Split a command line into its components and return them as a Python sequence.

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 on page 121 describes the functions you can use.

Table 40. Version information functions

Function	Description
`getApiMinorVersion`	Return the minor version of the API.
`getApiVersion`	Return the major version of the API.
`getTADDMVersion`	Return the TADDM version number.

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

```
getApMinoriVersion
```

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 functions 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 getLocalDNSLookup 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, which is described in "TADDM Javadoc information" on page 161.

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

target
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

mask
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

`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

`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:

```
import sys
import java

from java.lang import System
coll_home = System.getProperty("com.collation.home")
```

```
System.setProperty("jython.home",coll_home +
"/osgi/plugins/com.ibm.cdb.core.jython_1.0.0/lib")
System.setProperty("python.home",coll_home +
"/osgi/plugins/com.ibm.cdb.core.jython_1.0.0/lib")

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.

```
from java.lang import System
coll_home = System.getProperty("com.collation.home")

System.setProperty("jython.home",coll_home +
"/osgi/plugins/com.ibm.cdb.core.jython_1.0.0/lib")
System.setProperty("python.home",coll_home +
"/osgi/plugins/com.ibm.cdb.core.jython_1.0.0/lib")

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, **`sensorhelper.executeCommand`**(“echo hello world”).

```
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 sensorhelper.init() call.

```
def LogError(msg):
        log.error(msg)
        (ErrorType, 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.

```
   (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.

Typically, a custom server extension must run one or more commands or capture one or more files (or a combination of the two) to discover the intended target.

```
   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 `sensorhelper.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 four categories of TADDM database views:

- Building block views
- Details panel views
- Custom views
- Extended attributes views

Note: Fix Pack 3 In TADDM 7.3.0.3, and later, the maximum number of characters that column names in database views can have is 30.

Note: Fix Pack 5 A new view "MSS_INFO_VIEW" has been defined for the user to query the MSS Info details. User can write a query on this view to fetch the MSS Info specific data. To see the columns that are defined for this view, user can run the following SQL query:

```
DESCRIBE TABLE MSS_INFO_VIEW
```

Related concepts

"TADDM Data Dictionary" on page 159
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.

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 145.

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 an SQL **`describe`** command: Describe command returns multiple columns including the names of database views, which you can use to query the data displayed in the **Details Panel** tab.

```
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:

```
-- Column.... PK__OSRUNNING_C
-- Attribute..................... OSRunning
-- Java Type..................... model.topology.sys.OperatingSystem, notContained
-- Declared By................... model.topology.sys.ComputerSystem
```

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.

The time stamp column in TADDM 7.3.0.3, and later

Fix Pack 3

In TADDM 7.3.0.3, or later, the building block views contain additional time stamp column. For example, the view definition section of the LASTSTOREDTIME_C column of the BB_WINDOWSCOMPUTERSYSTEM20_V view contains additional (`timestamp`) string:

```
-- Column.... LASTSTOREDTIME_C
--            Attribute..................... lastStoredTime
--            Java Type..................... long, primitive (timestamp)
--            Declared By................... model.ModelObject
```

Each time stamp attribute contains the following two columns:

- LASTSTOREDTIME_C, which is set to `long` type, for example `1445417251307`.
- LASTSTOREDTIME_T, which contains human readable time stamp, for example `Oct 21, 2015 10:47:31 AM`.

Like in case of the original naming convention of the time stamp column, the added column always has the same suffix, which in this case is _T.

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:

```
-- 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_COMPUTERSYSTTEMS_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_COMPUTERSYSTTEMS_88841D4BJ mapping table).

Deprecated views

Some of the building block views become deprecated after changes in Tivoli Common Data Model (CDM) and will be deleted in the future. Refer to the table with the new equivalents of deprecated views to make necessary changes.

To make the reports that are already defined work after the upgrade, compatibility views are defined and marked as deprecated.

If you use any of the views that are listed in the following table, correct your report definition to use new equivalent view definition.

The following table lists deprecated views that support "many-to-many" relationships between object types, and their new equivalents.

Table 41. Deprecated views and their new equivalents.

Deprecated view	New equivalent
BB_APPCONFIGJDONCES_FCB57E09J	BB_SPHYSICALFILNCES_ECF04BA6J
BB_APPCONFIGJDO_ROLES_J	BB_SPHYSICALFILEJDO_ROLES_J
BB_APPSERVERCLUNCES_E03E73D6J	BB_SGROUPJDO_SENCES_CF68C060J
BB_APPSERVERCLUOLES_F1CF6FA2J	BB_SGROUPJDO_ROLES_J
BB_APPSERVERJDONCES_EF1C0CA8J	BB_SSOFTWARESERNCES_19E863EFJ
BB_APPSERVERJDO_ROLES_J	BB_SSOFTWARESERVERJDO_ROLES_J
BB_COLLECTIONJDNCES_2AB18ECEJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_COLLECTIONJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_COMPOSITEJDONCES_C1981AC5J	BB_SGROUPJDO_SENCES_CF68C060J
BB_COMPOSITEJDO_MEMBERS_J	BB_COLLECTIONJDO_MEMBERS_J
BB_COMPOSITEJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_COMPUTERSYSTEMJDO_ROLES_J	BB_SCOMPUTERSYSTEMJDO_ROLES_J

Table 41. Deprecated views and their new equivalents. (continued)	
Deprecated view	**New equivalent**
BB_COMPUTERSYSTNCES_611B3FC2J	BB_SCOMPUTERSYSNCES_119A868FJ
BB_COMPUTERSYSTNCES_6A2E6F7CJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_COMPUTERSYSTOLES_6B483FBCJ	BB_SGROUPJDO_ROLES_J
BB_CONFIGURATIONCES_F7B28A51J	BB_SFUNCTIONJDONCES_F8394E61J
BB_CONFIGURATIOOLES_33A89807J	BB_SFUNCTIONJDO_ROLES_J
BB_DB2DATABASEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_DB2DATABASEJNCES_83C5253DJ	BB_SDEPLOYABLECNCES_572F3E83J
BB_DB2SYSTEMJDONCES_6CAED009J	BB_SSOFTWAREINSNCES_549D08D8J
BB_DB2SYSTEMJDO_ROLES_J	BB_SSOFTWAREINSOLES_7C7BE7E0J
BB_DOMINOCONNECNCES_FAE09606J	BB_SPHYSICALFILNCES_ECF04BA6J
BB_DOMINOCONNECOLES_F12CCB72J	BB_SPHYSICALFILEJDO_ROLES_J
BB_DOMINODATABANCES_BE00AD89J	BB_SDEPLOYABLECNCES_572F3E83J
BB_DOMINODATABASEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_DOMINODOMAINJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_DOMINODOMAINNCES_AC5777E0J	BB_SGROUPJDO_SENCES_CF68C060J
BB_EXCHANGEADMINCES_6B41ACDEJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_EXCHANGEADMIOLES_5F958B9AJ	BB_SGROUPJDO_ROLES_J
BB_EXCHANGEFOLDNCES_EC2EEA5DJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_EXCHANGEFOLDOLES_B3A7C77BJ	BB_SGROUPJDO_ROLES_J
BB_EXCHANGEMAILNCES_2B5725CJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_EXCHANGEMAILOLES_250648DCJ	BB_SGROUPJDO_ROLES_J
BB_FABRICJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_FABRICJDO_SENCES_783F8B27J	BB_SGROUPJDO_SENCES_CF68C060J
BB_FUNCTIONJDO_NCES_C03DA9D4J	BB_SFUNCTIONJDONCES_F8394E61J
BB_FUNCTIONJDO_ROLES_J	BB_SFUNCTIONJDO_ROLES_J
BB_HACMPAPPRESONCES_B47F5A8AJ	BB_SFUNCTIONJDONCES_F8394E61J
BB_HACMPAPPRESOOLES_229BB16EJ	BB_SFUNCTIONJDO_ROLES_J
BB_HACMPLOCALREENTS_F67E36ECJ	BB_ITSYSTEMJDO_COMPONENTS_J
BB_HACMPLOCALRENCES_8ADCB6D9J	BB_SGROUPJDO_SENCES_CF68C060J
BB_HACMPLOCALREOLES_3C1CF07FJ	BB_SGROUPJDO_ROLES_J
BB_HACMPLOCALREOUPS_354DD1CCJ	BB_SERVICEGROUPOUPS_82AA2EE3J
BB_HACMPLOCALREROUP_21D3AC87J	BB_SERVICEGROUPOUPS_76D12CEDJ
BB_HACMPNODEJDONCES_F0DF78FDJ	BB_SFUNCTIONJDONCES_F8394E61J
BB_HACMPNODEJDO_ROLES_J	BB_SFUNCTIONJDO_ROLES_J

Table 41. Deprecated views and their new equivalents. (continued)

Deprecated view	New equivalent
BB_HIRDBSYSTEMJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_HIRDBSYSTEMJNCES_8CDFFAEEJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_IDSDATABASEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_IDSDATABASEJNCES_CFB10C79J	BB_SDEPLOYABLECNCES_572F3E83J
BB_IPNETWORKJDONCES_E3F0C245J	BB_SGROUPJDO_SENCES_CF68C060J
BB_IPNETWORKJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_ITSYSTEMJDO_NCES_31C6E3D2J	BB_SGROUPJDO_SENCES_CF68C060J
BB_ITSYSTEMJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_J2EEDOMAINJDNCES_111AC7A0J	BB_SGROUPJDO_SENCES_CF68C060J
BB_J2EEDOMAINJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_LINKSERVICEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_LINKSERVICEJNCES_F441B071J	BB_SDEPLOYABLECNCES_572F3E83J
BB_LOGICALCONTENCES_F200987CJ	BB_SPHYSICALFILNCES_ECF04BA6J
BB_LOGICALCONTENTJDO_ROLES_J	BB_SPHYSICALFILEJDO_ROLES_J
BB_MBEXECUTIONGNCES_A5DA5D10J	BB_SGROUPJDO_SENCES_CF68C060J
BB_MBEXECUTIONGOLES_6D4906A8J	BB_SGROUPJDO_ROLES_J
BB_MBMESSAGEFLONCES_2C1D04EAJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_MBMESSAGEFLONCES_EB919DCCJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_MBMESSAGEFLOOLES_18E0C30EJ	BB_SGROUPJDO_ROLES_J
BB_MBMESSAGEFLOWJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_MQINSTALLABLNCES_EDA0F908J	BB_SGROUPJDO_SENCES_CF68C060J
BB_MQINSTALLABLOLES_688C35B0J	BB_SGROUPJDO_ROLES_J
BB_MQQUEUEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_MQQUEUEJDO_SNCES_CAE5631FJ	BB_SDEPLOYABLECNCES_572F3E83J
BB_MSCLUSTERJDONCES_61127DF8J	BB_SGROUPJDO_SENCES_CF68C060J
BB_MSCLUSTERJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_MSCLUSTERNODEJDO_ROLES_J	BB_SFUNCTIONJDO_ROLES_J
BB_MSCLUSTERNODNCES_3899EF16J	BB_SFUNCTIONJDONCES_F8394E61J
BB_MSCLUSTERRESNCES_22175CCFJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_MSCLUSTERRESNCES_A728F28AJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_MSCLUSTERRESOLES_324E6849J	BB_SGROUPJDO_ROLES_J
BB_MSCLUSTERRESOLES_3A01196EJ	BB_SGROUPJDO_ROLES_J
BB_NETWORKSERVICEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_NETWORKSERVINCES_CDD7C865J	BB_SDEPLOYABLECNCES_572F3E83J

Table 41. Deprecated views and their new equivalents. (continued)

Deprecated view	New equivalent
BB_ORACLEDATABANCES_98E1A333J	BB_SDEPLOYABLECNCES_572F3E83J
BB_ORACLEDATABASEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_ORACLELISTENERJDO_ROLES_J	BB_SSOFTWARESERVERJDO_ROLES_J
BB_ORACLELISTENNCES_72725D9AJ	BB_SSOFTWARESERNCES_19E863EFJ
BB_ORACLESERVERJDO_ROLES_J	BB_SSOFTWAREINSOLES_7C7BE7E0J
BB_ORACLESERVERNCES_6F134AEBJ	BB_SSOFTWAREINSNCES_549D08D8J
BB_REALSERVERGRNCES_91B57D2EJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_REALSERVERGROUPJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_REALSERVERJDNCES_8B01D2CBJ	BB_SSOFTWARESERNCES_19E863EFJ
BB_REALSERVERJDO_ROLES_J	BB_SSOFTWARESERVERJDO_ROLES_J
BB_SAMETIMESERVERJDO_ROLES_J	BB_SFUNCTIONJDO_ROLES_J
BB_SAMETIMESERVNCES_7FDA5716J	BB_SFUNCTIONJDONCES_F8394E61J
BB_SEGMENTJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_SEGMENTJDO_SNCES_CBC65999J	BB_SGROUPJDO_SENCES_CF68C060J
BB_SERVICEGROUPROUP_6F30099EJ	BB_SERVICEGROUPOUPS_76D12CEDJ
BB_SERVICEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_SERVICEJDO_SNCES_63DE5757J	BB_SDEPLOYABLECNCES_572F3E83J
BB_SMSHIERARCHYJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_SMSHIERARCHYNCES_C6234B50J	BB_SGROUPJDO_SENCES_CF68C060J
BB_SMSSITECOMPONCES_FC696136J	BB_SDEPLOYABLECNCES_572F3E83J
BB_SMSSITECOMPOOLES_37331E42J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_SOFTWARECOMPNCES_5E176596J	BB_SDEPLOYABLECNCES_572F3E83J
BB_SOFTWARECOMPOLES_AA1C15E2J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_SOFTWAREINSTNCES_EECCCFCBJ	BB_SSOFTWAREINSNCES_549D08D8J
BB_SOFTWAREINSTOLES_B502FACDJ	BB_SSOFTWAREINSOLES_7C7BE7E0J
BB_SOFTWAREMODULEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_SOFTWAREMODUNCES_1FEAE999J	BB_SDEPLOYABLECNCES_572F3E83J
BB_SPECIALITYSENCES_A28738B4J	BB_SFUNCTIONJDONCES_F8394E61J
BB_SPECIALITYSEOLES_33910984J	BB_SFUNCTIONJDO_ROLES_J
BB_SQLSERVERDATNCES_4C800F20J	BB_SDEPLOYABLECNCES_572F3E83J
BB_SQLSERVERDATOLES_E33DFE98J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_STORAGEEXTENNCES_614C0287J	BB_SDEPLOYABLECNCES_572F3E83J
BB_STORAGEEXTENTJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_STORAGEPOOLJDO_ROLES_J	BB_SGROUPJDO_ROLES_J

Table 41. Deprecated views and their new equivalents. (continued)	
Deprecated view	**New equivalent**
BB_STORAGEPOOLJNCES_AC507A15J	BB_SGROUPJDO_SENCES_CF68C060J
BB_SYBASEDATABANCES_DFF7B51AJ	BB_SDEPLOYABLECNCES_572F3E83J
BB_SYBASEDATABASEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_SYSPLEXJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_SYSPLEXJDO_SNCES_4B3FE470J	BB_SGROUPJDO_SENCES_CF68C060J
BB_VCSSYSTEMJDONCES_831828D7J	BB_SGROUPJDO_SENCES_CF68C060J
BB_VCSSYSTEMJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_WEBLOGICMACHINEJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_WEBLOGICMACHNCES_41F6228FJ	BB_SGROUPJDO_SENCES_CF68C060J
BB_WEBLOGICNODENCES_1032390BJ	BB_SSOFTWARESERNCES_19E863EFJ
BB_WEBLOGICNODEOLES_14E2D98DJ	BB_SSOFTWARESERVERJDO_ROLES_J
BB_WEBSPHERENODEJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_WEBSPHERENODNCES_182E84E9J	BB_SGROUPJDO_SENCES_CF68C060J
BB_WINDOWSSERVICEJDO_ROLES_J	BB_SDEPLOYABLECOLES_C2D28F15J
BB_WINDOWSSERVINCES_9F30EF3AJ	BB_SDEPLOYABLECNCES_572F3E83J
BB_ZONEJDO_ROLES_J	BB_SGROUPJDO_ROLES_J
BB_ZONEJDO_SERVICEINSTANCES_J	BB_SGROUPJDO_SENCES_CF68C060J

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:

```
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:

```
########## DB2 Instance...<Layout> ##########
...General.....................................................<Tab Level 1>
......General..................................................<TabData>
......Db2Instance.General......................................<Content>
.........DP_DB2_INSTANCE_GENERAL_V.............................<View>
...System......................................................<Tab Level 1>
......System Info..............................................<TabData>
......Db2Instance.SystemInfo...................................<Content>
.........DP_DB2_INSTANCE_SYSTEM_INFO_V.........................<View>
......Profile Registry.........................................<TabData>
......Db2Instance.GlobalProfileRegistry........................<Content>
.........DP_DB2_INSTANCE_GLOB_PROFREG_V........................<View>
......License Info.............................................<TabData>
......Db2Instance.LicenseInfo..................................<Content>
.........DP_DB2_INSTANCE_LICENSE_INFO_V........................<View>
...Configuration...............................................<Tab Level 1>
......Configuration............................................<TabData>
......Db2Instance.Configuration................................<Content>
.........DP_DB2_INSTANCE_CONFIG_V..............................<View>
...Profile Registry............................................<Tab Level 1>
......Profile Registry.........................................<TabData>
......Db2Instance.ProfileRegistry..............................<Content>
.........DP_DB2_INSTANCE_PROFILE_REG_V.........................<View>
...Databases...................................................<Tab Level 1>
......Databases................................................<TabData>
......Db2Instance.Databases....................................<Content>
.........DP_DB2_INSTANCE_DATABASES_V...........................<View>
...Remote Databases............................................<Tab Level 1>
......Remote Databases.........................................<TabData>
......Db2Instance.Db2Alias.....................................<Content>
.........DP_DB2_INSTANCE_DB2_ALIAS_V...........................<View>
...Modules.....................................................<Tab Level 1>
......Software Modules.........................................<TabData>
......AppServer.SoftwareModules................................<Content>
.........DP_APP_SERVER_SW_MODULES_V............................<View>
......Other Modules............................................<TabData>
......AppServer.StaticModules..................................<Content>
.........DP_APP_SERVER_STATIC_MOD_V............................<View>
...Application Descriptors.....................................<Tab Level 1>
......Application Descriptors..................................<TabData>
......AppServer.ApplicationDescriptors.........................<Content>
.........DP_APP_SERVER_APP_DESCRIPS_V..........................<View>
...Runtime.....................................................<Tab Level 1>
......Databases................................................<TabData>
......Db2Instance.Runtime......................................<Content>
.........DP_DB2_INSTANCE_RUNTIME_V.............................<View>
```

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_GENERAL_V view and filtering by the instance name (see “Example query” on page 152).

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 file. The following comment block provides information about the nodes, or CDM object types, referred to by the DP_DB2_INSTANCE_DATABASES_V view:

```
-- ######## Db2Instance.Databases ########
--
-- View...... DP_DB2_INSTANCE_DATABASES_V
--
-- Node...... 1
-- Node Path..................... Db2Instance
-- Node Class Name............... model.topology.app.db.db2.Db2Instance
-- Node Type..................... Root
-- Node...... 2
-- Node Path..................... Db2Instance._arraydatabases
-- Node Class Name............... model.topology.app.db.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.db.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:

```
db2 "describe table DP_DB2_INSTANCE_DATABASES_V"

Column                         Type      Type
name                           schema    name                Length   Scale Nulls
------------------------------ --------- ------------------ -------- ----- ------
NAME_C3                        SYSIBM    VARCHAR                 192     0 Yes
ALIAS_C3                       SYSIBM    VARCHAR                 192     0 Yes
PK_C3                          SYSIBM    VARCHAR                 192     0 Yes
PK_C1                          SYSIBM    VARCHAR                 192     0 No
```

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:

```
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.

Element	Attributes	Contained elements
`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`
`plain`	**`fieldName`** The field name of the plain **`nameInView`** The name of the column to expose in the database view. The maximum length is 30 characters. Avoid DB2 or Oracle reserved words. **`displayType`** The type of value. This attribute is optional. Specify one of the following values: **`speed`** A value in MHz **`memory`** A value in KB, MB, or GB **`mBytes`** A value in MB **`date`** A timestamp in `YYYY-MM-DD-HH24:MI:SS` format, used for fields containing epoch time in milliseconds **`networkSpeed`** A value in megabits per second **`StorageGBytes`** A value in GB	None

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:

```
<plain fieldName="fqdn" nameInView="FDQN"/>
```

- 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:

```
<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:

```
<views>
  <!-- ComputerSystems with OS, Filesystems -->
  <view className="com.collation.platform.model.topology.sys.ComputerSystem"
  viewName="CM_COMPUTER_SYSTEMS_V" includePrimaryKeys="false">
    <field>
      <plain fieldName="fqdn" nameInView="FDQN"/>
    </field>
    <field>
      <nested className="com.collation.platform.model.topology.net.IpInterface"
      fieldName="ipInterfaces">
        <nested className="com.collation.platform.model.topology.net.IpAddress"
        fieldName="ipAddress">
          <plain fieldName="dotNotation" nameInView="IP_ADDRESS"/>
          <plain fieldName="stringNotation" nameInView="IP_ADDRESS_STRING"/>
        </nested>
      </nested>
    </field>
    <field>
      <nested className="com.collation.platform.model.topology.sys.OperatingSystem"
      fieldName="OSRunning">
        <plain fieldName="OSName" nameInView="OS_NAME"/>
      </nested>
    </field>
    <field>
      <plain fieldName="CPUType" nameInView="CPU_TYPE"/>
      <plain fieldName="numCPUs" nameInView="NUM_CPUS"/>
      <plain displayType="speed" fieldName="CPUSpeed" nameInView="CPU_SPEED"/>
      <plain displayType="memory" fieldName="memorySize" nameInView="MEMORY_SIZE"/>
      <plain fieldName="primaryMACAddress" nameInView="PRIMARY_MAC_ADDRESS"/>
      <plain fieldName="serialNumber" nameInView="SERIAL_NUMBER"/>
      <plain fieldName="manufacturer" nameInView="MANUFACTURER"/>
      <plain fieldName="model" nameInView="MODEL"/>
      <plain fieldName="type" nameInView="SYSTEM_TYPE"/>
    </field>
    <field>
      <nested className="com.collation.platform.model.topology.sys.FileSystem"
      fieldName="fileSystems">
        <plain displayType="mBytes" fieldName="capacity" nameInView="CAPACITY"/>
        <plain displayType="mBytes" fieldName="availableSpace"
        nameInView="AVAILABLE_SPACE"/>
        <plain fieldName="displayName" nameInView="FILE_SYSTEM"/>
      </nested>
    </field>
  </view>

  <!-- AppServers with host -->
  <view className="com.collation.platform.model.topology.app.AppServer"
  viewName="CM_APP_SERVERS_PER_HOST_V" includePrimaryKeys="false">
    <field>
      <nested className="com.collation.platform.model.topology.sys.ComputerSystem"
      fieldName="host">
        <plain fieldName="fqdn" nameInView="FQDN"/>
        <nested className="com.collation.platform.model.topology.net.IpInterface"
        fieldName="ipInterfaces">
          <nested className="com.collation.platform.model.topology.net.IpAddress"
          fieldName="ipAddress">
            <plain fieldName="dotNotation" nameInView="IP_ADDRESS"/>
            <plain fieldName="stringNotation" nameInView="IP_ADDRESS_STRING"/>
          </nested>
        </nested>
      </nested>
    </field>
```

```
    <field>
      <plain fieldName="jdoClassName" nameInView="CLASS_NAME"/>
      <plain fieldName="objectType" nameInView="TYPE"/>
      <plain fieldName="productName" nameInView="PRODUCT_NAME"/>
      <plain fieldName="productVersion" nameInView="PRODUCT_VERSION"/>
      <plain fieldName="keyName" nameInView="KEY_NAME"/>
      <plain fieldName="name" nameInView="NAME"/>
    </field>
  </view>
</views>
```

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`
   - `drop_custom_views_db2.sql`
   - `drop_custom_views_oracle.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`

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 two extended attribute views:

- 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.
- Fix Pack 1 BE_*model*_V that has columns corresponding to the Common Data Model attributes and to the extended attributes. When the column name conflict occurs, the extended attribute columns might not be present.

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 modify an extended attribute, you must drop the existing view before you use the extended attribute view tool to create an updated SQL script and an updated attribute view.

Data type modification

Fix Pack 3

In TADDM 7.3.0.2, and earlier, the `extattr_views.sh` script creates extended attributes views with columns of type CLOB. Starting with 7.3.0.3, the views contain data of specific types, for example, VARCHAR, or SMALLINT. The following table provides the new column types in the extended attributes views in DB2 and Oracle databases.

Table 42. Extended attributes views column types in DB2 and Oracle databases

GUI type	Column type in DB2	Column type in Oracle
String	VARCHAR(32000)	VARCHAR2(4000)
Character	VARCHAR(4)	VARCHAR2(4)
Double precision floating point	DOUBLE	NUMBER
Floating point	REAL	NUMBER
Boolean	SMALLINT	NUMBER(1)
Integer	INTEGER	NUMBER
Short integer	SMALLINT	NUMBER
Long integer	BIGINT	NUMBER

Such modification might affect integration with products that use TADDM database views for extended attributes. For example, if you use BIRT reports, errors might be generated if extended attributes columns are not cast to a specific data type, for example, VARCHAR.

Note: In TADDM 7.3.0.2, and earlier, extended attributes of the boolean type are displayed as `false`, or `true` in the database views. In TADDM 7.3.0.3, and later, the value is an integer, either `0`, or `1`. It means that the BIRT and Cognos reports that were created before this change are not compatible with the new data types.

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

`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 two views:

- `EA_COMPUTERSYSTEM40_V`
  The new view has the following columns:
  - `PK_C`, which contains the primary key.
  - `SUPPORT_AREA_C`, which contains the extended attributes.
- Fix Pack 1 `BE_COMPUTERSYSTEM40_V`
  The new view has the following columns:
  - All columns from the building block view `BB_COMPUTERSYSTEM40_V`, for example `PK_C`, `NAME_C`, `VMID_C`.
  - `SUPPORT_AREA_C`, which contains the extended attributes, only if there is no column name conflict with the columns from the building block view `BB_COMPUTERSYSTEM40_V`.

You can use the following SQL commands to query the data:

```
SELECT
         T1.FQDN_C,
         T2.SUPPORT_AREA_C
     FROM
        BB_COMPUTERSYSTEM40_V T1,
        EA_COMPUTERSYSTEM40_V T2
```

```
    WHERE
       T1.PK_C = T2.PK_C
```

Fix Pack 1
```
SELECT
    FQDN_C, SUPPORT_AREA_C
FROM
    BE_COMPUTERSYSTEM40_V
```

Note: Fix Pack 1 If a column name conflict occurs, the tool tries to add extended attribute category string. If this does not resolve the problem, the extended attribute column is not included in the `BE_model_V` views.

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`*`:`*`port`*`/cdm/datadictionary/`, for example `http://1.123.123.12:9430/cdm/datadictionary/`
- In the `taddm-data-dictionary.zip` file that is in the `$COLLATION_HOME/sdk/datadictionary` and `$COLLATION_HOME/deploy-tomcat/cdm/datadictionary` (TADDM 7.3.0), or `$COLLATION_HOME/apps/cdm/datadictionary` (TADDM 7.3.0.1, and later) 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 `$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:

```
http://taddmserverhost:port/cdm/datadictionary/bb-views/index.html
```

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:

```
http://taddmserverhost:port/cdm/datadictionary/model-object/index.html
```

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:

```
http://taddmserverhost:port/cdm/datadictionary/cdm-tables/index.html
```

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:

```
http://taddmserverhost:port/cdm/datadictionary/sensors/index.html
```

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:

```
http://taddmserverhost:port/cdm/datadictionary/sensors/potentialData.html
```

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

Additional indexes

In the `datadictionary/cdm` directory of Data Dictionary you can also find the following indexes:

- Class Index at `http://taddmserverhost:port/cdm/datadictionary/cdm/classes/$index.htm`
- Interface Index at `http://taddmserverhost:port/cdm/datadictionary/cdm/interfaces/$index.htm`
- Attribute Index at `http://taddmserverhost:port/cdm/datadictionary/cdm/attributes/$index.htm`
- Relationship Index at `http://taddmserverhost:port/cdm/datadictionary/cdm/relationships/$index.htm`
- Datatype Index at `http://taddmserverhost:port/cdm/datadictionary/cdm/datatypes/$index.htm`

- Naming Rule Index at `http://taddmserverhost:port/cdm/datadictionary/cdm/namingrules/$index.htm`

TADDM Javadoc information

You can used the Javadoc information that is included with TADDM to find out more about the APIs that are available.

The following compressed files contain TADDM Javadoc information:

$COLLATION_HOME/sdk/doc/api/oalapi-javadoc.zip
This file contains information about the available TADDM Java API.

$COLLATION_HOME/sdk/doc/api/taddmapi-javadoc.zip
This file contains information about the available TADDM Java API.

$COLLATION_HOME/sdk/doc/capabilities/capabilities-javadoc.zip
This file contains information about the capabilities functionality that you can use.

$COLLATION_HOME/sdk/doc/model/model-javadoc.zip
This file contains information about the Common Data Model objects used by TADDM.

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.

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 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.