Tivoli Application Dependency Discovery Manager
Version 7.3

Discovery Library Adapter Developer's Guide

Note

Before using this information and the product it supports, read the information in “Notices” on page 19.

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. Discovery Library Adapter Developer's Guide

Using Discovery Library Adapters

The Discovery Library provides an integration mechanism for communicating and sharing information about discovered resources and relationships within the enterprise.

The Discovery Library consists of the following components:

Discovery Library XML schema specification
This schema is called the Identity Markup Language (IdML), which defines a set of operations for creating, updating, and deleting objects in the Common Data Model (CDM).

Discovery Library Adapter (DLA)
DLAs are application code written to extract discovered resource and relationship data, and then transformed to the IdML specification.

Discovery Library books
These are XML files formatted according to the IdML that contain discovery information, including the identity of resources and their relationships.

Discovery Library File Store (DLFS)
This is a repository for Discovery Library books.

The following sequence describes the Discovery Library information flow:

1. A Management Software System (MSS) discovers resources and relationships in an enterprise environment.
2. A DLA creates an IdML representation of the MSS application data (resources and relationships). The DLA can also request discovery updates, as required.
3. The DLA copies the IdML book to the DLFS enabling readers, such as the bulk load program, to access the resource and relationship information.

Discovery Library Adapter overview

A Discovery Library Adapter is a runtime component in the Discovery Library that uses mechanisms in Management Software Systems (MSS) to extract specific details about resources and resource relationships. The purpose of Discovery Library Adapters is to discover and maintain the resources and resource relationships that support business applications.

Discovery Library Adapters transform this information into files that conform to the Identity Markup Language (IdML) schema and store the resulting IdML books in the Discovery Library File Store.

IdML schema representation

IdML is the Discovery Library XML schema specification. Discovery Library Adapters output files in IdML format, which contain information about the Management Software System (MSS) and operation sets that define groups of operations for creating, updating, and deleting objects in the Common Data Model (CDM).

The IdML schema references the Common Data Model schema, which describes CDM model objects and relationships and their corresponding representations in XML format.

IdML books

IdML books, also known as Discovery Library books, are XML files that contain information about resources and resource relationships written to conform to the IdML schema. Each IdML book represents the distinct view of the resources and relationships at a point in time. Collections of IdML books will

therefore often represent overlapping views of the environment. Readers of IdML books are therefore responsible for merging these views into a consistent whole that is meaningful in the context of the application.

IdML books uniquely identify the author of the discovery data. An IdML book can describe either delta or complete (also known as refresh) discoveries. See the following section for more information. The Discovery Library provides an application programming interface (API) for the creation of well-formed IdML books, so Discovery Library Adapter developers can focus on the extraction and transformation of data.

IdML operation semantics

Operation sets stored in IdML books can represent the following semantics:

Delta
Operations in the IdML book represent changes and updates to existing data imported during previous runs of IdML books for a particular Management Software System.

Refresh
Operations in the IdML book represent a refresh operation of existing data imported from previous runs of IdML books for a particular Management Software System. Resources present from prior runs but not present in the refresh operation are removed. Refresh files represent a snapshot in time, replacing existing information with new data.

Discovery Library File Store

A Discovery Library File Store (DLFS) is a repository for Discovery Library books (IdML books). A Discovery Library File Store can reside on a local system or can be accessible through a network connection. After a Discovery Library Adapter writes a book to a Discovery Library File Store, the book should not be modified.

IdML schema

The discovery library uses an XML format called Identity Markup Language (IdML) to enable data collection. Access to the IdML code is provided through the discovery library adapters.

The XML Schema Definition (XSD) describes the operations that are necessary to take data about resource and relationship instances from an author and instantiate it into the repository of a reader. This schema defines the operations that occur on instances of resources and relationships. To facilitate future model versions and updates, this schema references an external schema, the Common Data Model, to define the resource and relationships. All files that are in the Discovery Library conform to the IdML schema. Books in the Discovery Library that do not validate against the IdML schema are in error and cannot be used by readers. TADDM is an example of a reader.

The IdML schema is designed to separate the operations from the model specification to enable the schema to handle updates to the model specification without changing the IdML schema. The TADDM reader treats the individual elements within the operations as a transaction.

TADDM XML and IdML

There are some differences between IdML files and TADDM XML files. The following table summarizes the differences.

Table 1. Differences between IdML books and TADDM XML files

IdML books	**TADDM XML files**
IdML is a standard that supports objects defined in the Common Data Model. ACLs, users, scopes, and schedules are not supported.	TADDM XML files is an application format that supports objects defined in the Common Data Model and TADDM. ACLs, users, scopes, and schedules are supported.

Table 1. Differences between IdML books and TADDM XML files (continued)	
IdML books	**TADDM XML files**
Operation codes include delta (default), and refresh.	There are no operation codes. The delta behavior is the default.
In-file MSS information is supported.	MSS information can be provided through the command line.
Relationships have to be defined explicitly.	Implicit relationships can and must also be defined.
XML objects are not nested.	XML objects are nested.
Virtual, relative IDs are supported. Relationships can link configuration items defined with relative IDs.	Relative IDs are not supported. Real objectGUID is supported. Relationships must link configuration items identified with GUID or naming attributes.

When to use a Discovery Library Adapter

Discovery Library Adapters offer specific advantages to help you integrate enterprise information with the IBM TADDM database. Learn when a DLA is most useful.

Consider creating a DLA in the following cases:

- A discovery tool exists that can create a data file containing discovered resources and relationships.
- The solution requires a loose integration with existing management technology.
- The environment demands a quick integration solution.
- There is a need to use an existing discovery scheduler.
- There is a requirement to minimize native environment interruptions.

Alternatively, consider using the TADDM API as an integration solution in the following cases:

- The environment requires real time storage of information in the TADDM database.
- The solution would benefit from making interactive calls to the TADDM database to store information.
- The system requires synchronous acknowledgement that creates, updates, and deletes have completed successfully in the TADDM database.
- There is a need to reduce the overhead and delay of processing books and data.

In general, the TADDM APIs provide more timely, synchronous, programmatic-style integration. DLAs provide more loosely coupled, asynchronous implementations, offering greater flexibility in many environments. Using DLAs, you can also maintain a degree of technology independence from the TADDM implementation, including the TADDM API, the programming model, and specific runtime aspects.

File naming conventions

Identity Markup Language (IdML) books are stored in plain text XML files which must follow a consistent file naming convention. The file name includes information to uniquely identify the book within the Discovery Library File Store. This information helps developers and administrators to quickly identify the source and creation date of the discovery data.

IdML book names consist of the following segments:

- The application code of the Management Software System (MSS).

  Every Discovery Library Adapter requires an application code (10 character maximum). Include the short name of the application together with the version.
- The host name of the MSS.
- An ISO 8601 time stamp UTC (Coordinated Universal Time), with colons (:) replaced by dots (.).

- The text "refresh" when the book contains a refresh operation.
- A file name extension of `.xml`.

Sample file names

The following file name example is for an IdML book that is in the Discovery Library:

```
AppAv1.3.host.abcxyz.com.2006-03-07T12.05.00Z.xml
```

The following file name example is for an IdML book that is in the Discovery Library that contains a refresh operation:

```
AppAv1.3.host.abcxyz.com.2006-03-07T12.05.00Z.refresh.xml
```

Naming files while copying to a Discovery Library

There is a specific naming convention that only applies during the writing and copying of IdML books to the Discovery Library. In this case, the file name of the IdML book must contain the `.partial` suffix, as shown in the following example:

```
AppAv1.3.host.abcxyz.com.2006-03-07T12.05.00Z.xml.partial
```

After the file is written or copied to the Discovery Library File Store, rename the file name by removing the `.partial` suffix, as shown in the following example:

```
AppAv1.3.host.abcxyz.com.2006-03-07T12.05.00Z.xml
```

Integration overview

Integrating a Management Software System (MSS) with the TADDM database consists of two procedures, which are mapping the MSS data to the Common Data Model (CDM) and creating a DLA that implements the model mapping and generates an IdML book.

Mapping the MSS data to the CDM begins with collecting and analyzing the source data, as assembled by the MSS, with the intent of understanding the content and purpose of the information. After you do this, you can identify corresponding model objects within the CDM and determine how to apply the CDM naming rules to create unique instances of the data.

In the process of defining model objects, you can look for additional relationships between objects to capture maximum information about the environment. As part of model mapping, you must also verify that the MSS data is consistent with attribute conventions within the CDM.

Note: The MSS is responsible for performing discovery, monitoring resources, and capturing the data.

To create a DLA, use the model map that you created earlier to define a set of operations that creates, updates, and deletes data within the CDM. The output of the DLA is a well-formed IdML book, in XML format, that resides in the Discovery Library available to reader applications, such as the Bulk Load program.

Creating a Discovery Library Adapter

Learn how to create a Discovery Library Adapter, how to develop a model map using data from the Management Software System (MSS), and how to use the DLA API to build the Identity Markup Language (IdML) book.

To create a DLA, complete the following steps:

1. Collect a representative sample (complete, if possible) of the type of data generated by the MSS.
2. Determine the type of resources and relationships supplied by the MSS.

   It is important to understand the content and purpose of the data generated by the MSS. For each item, determine the following:

- Is the item a specific resource, a category of resources, or a relationship?
- How does the MSS use the item?
- How was the item discovered?

You can use this information to more accurately identify the type of model objects that should represent the item.

3. Identify model objects within the Common Data Model (CDM) corresponding to entities within the MSS data.

   Identifying model objects is part of the process of creating a mapping between the CDM and the MSS data. For example, if the MSS contains a data item with an attribute of Windows, you can begin by deriving the following information to represent this data item:

   - There is an operating system of type Microsoft Windows.
   - There is a host computer system.
   - There is a relationship between the operating system and the computer system.

   Continue this process of identifying model objects present within the MSS data, examining the targets of explicit and implied relationships. For instance, if there is a source data item representing a computer system, determine additional characteristics (such as the IP address) of the computer system represented in the data. Document information about the MSS data and potential model objects for future reference.

4. For each model object you identify, use the CDM naming rules to determine the attributes and relationships that are required to create an instance of a resource.

   CDM naming rules define a set of attributes and relationships that provide the necessary naming context to create a unique identity for a model object. Potentially, there are multiple naming rules for each model object; each object instance must use at least one of the rules for each object type when mapping application resource data.

   In some instances, the MSS data might not include enough information to provide unique identification of a resource. In this case, naming rules require that the resource be named not only with characteristics that are specific to it, but also with characteristics of the resource within the context of another instance.

   For example, one of the naming rules for operating system type and operating system name specifies that the naming context be an instance of a computer system. This means that in order to create a mapping to an operating system name and operating system type, you must also define an instance of the associated computer system, along with the relationship between the computer system and the operating system.

   Naming contexts are always specified in terms of other resources, relationships to other resources, or the attributes of other resources.

5. Apply CDM relationships between currently identified model objects, as appropriate.

   Refer to the UML diagram for the CDM to determine potential model object relationships. Note that relationships are hierarchical, which means that relationships between model objects are automatically valid for subclasses of the model objects. For example, a runsOn relationship between an operating system and a computer system is valid for operating system subclasses. You do not need to define explicit relationships that mirror the hierarchy of the CDM.

6. Verify that the MSS data is consistent with the attribute conventions used to store existing information in the CDM and reconcile them as necessary.

   Attribute content consistency is critical. When verifying attribute content consistency, consider the following points:

   - The format of the data, including the use of dashes, dots, and other delimiters
   - Whether special characters are present
   - Units of measure, if appropriate
   - Case sensitivity and whether the data is typically in upper, lower, or mixed case.

For example, consider the case of serial numbers for computer systems. One technology might require the use of only capital letters and dashes, while another technology might consider dashes to be restricted characters. Make note of any data processing requirements uncovered during the model mapping stage.

7. Create a model mapping document and map the data from the MSS to the CDM.

   Use the Data Model Template as a guide for creating this document. Completing the following steps to creating a model mapping document:

   a) Define a usage scenario.

      You must create at least one scenario to describe the data usage in the mapping document. The scenario helps you validate that you are gathering the necessary information from the MSS. For example, the MSS might be collecting information about operating systems, but you might also need to know about running instances of application servers.

      A sample scenario could read as follows: The instance data based on classes *x* and *y* enables *Application A* to automate application mapping in the provisioning module.

   b) List all CDM attribute content conventions.

   c) Specify the CDM classes (model objects) and associated attributes used.

   d) List the relationships provided in the MSS data.

   e) List CDM classes with their associated naming policy and naming rules.

8. Use the model mapping document to define operations and operation sets for creating, modifying, and deleting model objects (managed elements).

9. Use the DLA API to build the IdML book.

   The DLA API consists of an adapter API and a book generation API. You are not required to use the DLA API, but the production API offers considerable assistance in creating well-formed IdML books conforming to the IdML schema. Similarly, the adapter API offers common interfaces for command and control, and other operations such as starting and stopping discoveries. See "Understanding the DLA APIs" on page 8 for more information.

   As part of creating the IdML book, you must also assign a file name to the Discovery Library book. See "File naming conventions" on page 3 for complete information about Discovery Library book file naming conventions.

10. Save the book to the Discovery Library File Store by completing the following steps. You must have write and file rename permissions on the file store.

    a) Append a `.partial` suffix to the name of the book when saving it to the Discovery Library.

       Books copied or delivered to the Discovery Library File Store must include the `.partial` suffix during the copy operation, for example:

       ```
       APPAv1.1.host.abcxyz.com.2006-03-07T12.05.00Z.xml.partial
       ```

    b) After the book is completely written to the Discovery Library, remove the `.partial` suffix from the file name.

IBM Discovery Library IdML Certification tool

IBM Discovery Library IdML Certification tool reviews Discovery Library books. It verifies whether the content of the books complies with the IdML specification. If any errors are found, the certification tool reports the errors to the console.

The tool runs the following certification tasks:

- Certifies a book against the IdML and CDM schemas.
- Certifies that all managed elements specify a valid set of naming rule attributes, so that at least one valid naming rule is formed.
- Certifies relationships in the following ways:

  - Certifies that the source and target resources of each relationship reference an existing management element.
  - Certifies that all relationships reference valid source and target classes.
- Certifies that there are no missing superiors. If a management element references a superior, the superior management element must exist in the book.
- Certifies the attributes in the following ways:
  - Certifies that a managed element instance has at least one attribute.
  - Certifies that no attribute is empty.
- Provides statistics on the number of classes and relationships that are used.

Using IBM Discovery Library IdML Certification tool

To use the tool, complete the following steps:

1. Go to the `$COLLATION_HOME/sdk/dla/validator/v2` directory.
2. Run the following command:

```
java -jar idmlcert.jar <options>
```

Examples of the command options

- To display the usage information, run the following command:

```
java -jar idmlcert.jar -?
```

- To certify the `mytestfile.xml` file, which is in the current directory, run the following command:

```
java -jar idmlcert.jar mytestfile.xml
```

- To certify the files that are listed in the `books.lst` file, which is in the `/dla` directory, run the following command:

```
  java -jar idmlcert.jar -f /dla/books.lst
```

  where the `books.lst` file contains the following files:

  - `/dla/file1.xml`
  - `/dla/file2.xml`

- To certify the files that are listed in the `/dla/books.lst` file, which is in the current directory and, which uses the `idmlcert.properties` properties file, run the following command:

```
  java -jar idmlcert.jar -p idmlcert.properties
```

  where the `idmlcert.properties` file contains the following property:

```
com.ibm.dl.core.certification.bookListFilename=/dla/books.lst
```

- To certify the `bigBook.xml` book file, which has hundreds of megabytes, run the following command:

```
  java -Xmx2560m -jar idmlcert.jar bigBook.xml
```

  where the value of the `Xmx` option must be at least 4-6 times bigger than the size of the book.

See also *The bulk load program* and *Delta books utility program* topics is the TADDM *User's Guide*.

Understanding the DLA APIs

The Discovery Library provides the two application programming interfaces (API) to facilitate the integration of discovery data from a Management Software System into the Discovery Library, which are Adapter API and Book Production API.

Adapter API
Use this API to start and stop discoveries, as well as create transient or long-running Discovery Library Adapters. See "Using the DLA adapter API" on page 8 for more information.

Book Production API
Use this API to create well-formed IdML books conforming to the IdML schema. See "Using the DLA Book Production API" on page 12 for more information.

You can use the adapter and book production APIs either in conjunction or independently of each other.

Using the DLA adapter API

You can use the DLA adapter API with a Management Software System (MSS).

To use the APIs described in this document, make sure the `.jar` files in the `$COLLATION_HOME/sdk/dla/dla_utility` directory are in a location listed on the CLASSPATH of your system.

Each MSS typically has its own conventions and requirements regarding configuration, deployment, control, and security. You can use the DLA adapter API to develop discovery code that can interface with an MSS and be reused in different runtime environments. The adapter API is implemented through the DiscoveryLibraryAdapter abstract class that provides methods for all supported Discovery Library Adapter functions.

To create a DLA for an MSS, complete the following steps:

1. Extend the DiscoveryLibraryAdapter abstract class and override the implementations for the `getCapabilities` and `getConfigParams` class scope methods and the `getState` and `stopDiscovery` methods.
2. Provide implementations for the abstract `setConfigParams` and `startDiscovery` methods.

   These methods are described in "Managing configuration parameters and discoveries" on page 8.
3. Override the `addPropertyChangeListener` and `removePropertyChangeListener` methods.

   You are not required to override these methods since `addPropertyChangeListener` and `removePropertyChangeListener` are concrete methods in DiscoveryLibraryAdapter class. "Managing property change listeners" on page 11 describes the property change listener methods.
4. Implement the `start`, `pause`, `resume`, and `shutdown` methods for long-running DLAs.

   A DLA can run in either transient or long-running mode. A transient DLA can be thought of as running a one-time discovery, initializing, performing the discovery operation according to the configuration properties and, when completed, shutting down. Transient DLAs do not maintain an internal state that can be interrogated.

   A long-running DLA maintains an internal state over time, which you can control using the state manipulation methods including `start`, `pause`, `resume`, and `shutdown`. A long-running DLA can perform discoveries when it is in the *running* state through a call to the `startDiscovery` method. These methods for writing long-running DLAs are described in "Managing Discovery Library Adapter states" on page 11.

Managing configuration parameters and discoveries

You can use the methods described in this section to determine and set the configuration parameters required by an instance of a DLA. You can also use the methods to start and stop a discovery, and determine the state of the DLA. Using these parameters is optional for the creation of a DLA.

Table 2 on page 9 describes the methods for managing configuration parameters and discoveries.

Table 2. Configuration parameters and discovery methods	
Method	**Description**
`getCapabilities()`	This a class scope method that returns a set of properties indicating the capabilities of the DLA. These capabilities include the types of discoveries are supported, along with whether the DLA can support transient or long-running behaviors. By default this method returns null (indicating that the capabilities of the DLA are unknown).
`getConfigParams()`	This a class scope method that returns a structure specifying the configuration parameters required by an instance of the DLA. By default this method returns null (indicating that the configuration parameters of the Discovery Library Adapter are unknown).

Table 2. Configuration parameters and discovery methods (continued)

Method	Description
`getState()`	Retrieve the current state of the DLA. The state values for a DLA are: **0 (Stopped)** The DLA is not currently performing any function and will continue in this state until started using the `start()` method. By default, this is the initial state of a DLA. **1 (Starting)** The DLA is in the process of starting as a result of a call to the initialize or start methods. **2 (Running)** The DLA is running. This does not imply that a discovery is in progress, only that the module is active and can perform discoveries. **3 (Pausing)** The DLA is in the process of pausing as a result of a call to the `pause()` method. **4 (Paused)** The DLA is paused. You can use this state with DLAs that maintain internal state variables whose values persist across calls to the `pause()` and `resume()` methods. **5 (Resuming)** The DLA is in the process of resuming as a result of a call to the `resume()` method. **6 (Stopping)** The DLA is in the process of stopping as a result of a call to the `shutdown()` method. **7 (Recovering)** A DLA that should be running is attempting to recover from an internal error. **8 (Aborted)** The DLA ended abnormally and is not attempting to recover. **9 (Discovering)** The DLA is currently in the process of performing a discovery.
`setConfigParams(configParams)`	Initialize the API by providing a completed set of configuration properties that contain sufficient information for the API to connect to and extract data from its data sources.
`startDiscovery( )`	Start a discovery using the set of parameters specified using the `setConfigParams()` method. If successful, the DLA returns a value identifying the discovery that has been started.
`stopDiscovery(discoveryId)`	Stop a previously started discovery.

Managing property change listeners

The methods described in this section enable you to add and remove listeners for notification when DLA properties change. Using these parameters is optional for the creation of a DLA.

The DiscoveryLibraryAdapter class defines a single property, though you can add additional properties that support notification.

Table 3 on page 11 describes the methods for managing property change listeners.

Table 3. Property change listener methods

Method	Description
`addPropertyChangeListener(propertyName, listener)`	Register an object for notification when a DLA property changes.
`removePropertyChangeListener( propertyName, listener)`	Unregister an object from the list of those to be notified when a DLA property changes.

Managing Discovery Library Adapter states

You can use the methods described in this section to start, pause, resume, and shutdown a long-running DLA. Using these parameters is optional for the creation of a DLA.

Table 4 on page 11 describes the methods for managing the state of DLAs.

Table 4. State methods

Method	Description
`pause()`	Pause a long-running DLA. The return value is the state of the DLA after executing the call, either 3 (Pausing) or 4 (Paused) depending on whether the DLA is capable of immediately pausing. By default, this method returns the current value of the state attribute.
`resume()`	Resume a long-running DLA. The return value is the state of the DLA after executing the call, either 5 (Resuming) or 2 (Running) depending on whether the DLA is capable of immediately resuming from a paused state. By default, this method returns the current value of the state attribute.
`shutdown()`	Shut down a long-running DLA. The return value is the state of the DLA after executing the call, either 6 (Stopping) or 0 (Stopped) depending on whether the DLA is capable of immediately stopping from its current state. By default, this method returns the current value of the state attribute.

Table 4. State methods (continued)	
Method	**Description**
`start()`	Start a DLA that has already been Initialized The return value is the state of the DLA after executing call, either 1 (Starting) or 2 (Running) depending on whether the DLA is capable of immediately entering a running state from a stopped state. By default, this method returns the current value of the state attribute.

Using the DLA Book Production API

You can use the DLA book production API to create well-formed Identity Markup Language (IdML) books conforming to the IdML schema.

To create IdML books, complete the following steps:

1. Create an instance of the IDMLBook class and initialize it by calling the `create()`, `getBookName()`, and `openBook()` methods.

   See "Book properties and methods" on page 12 for more information.
2. Add operation sets to the book by calling the `openOperationSet()` and `closeOperationSet()` methods.

   See "Book properties and methods" on page 12 for more information.
3. Add operations to the operation sets by calling the `openCreateOperation()`, `openDeleteOperation()`, `openModifyOperation()`, and `openRefreshOperation()` methods.

   See "Book properties and methods" on page 12 for more information.
4. Within each operation, such as create or delete, add managed elements and relationships using the `addManagedElement()` and `addRelationship()` methods.

   See "Managed element properties and methods" on page 15 and "Relationship properties and methods" on page 17 for more information.
5. Call the appropriate close method to complete each operation and operationSet, as required.
6. Close the book and complete production by calling the `closeBook()` method.

Book properties and methods

You can use the book production properties and methods to open and close IdML books, define and specify operation sets and operations, and add managed elements and relationships to operations.

Properties

Table 5 on page 12 describes the book properties for the DLA production API.

Table 5. Book production properties	
Property	**Description**
`source`	The cdmManagementSoftwareSystem instance that identifies the source of the discovery data contained in the book.
`timestamp`	The IdML book creation time, specified using UTC.

Methods

Table 6 on page 13 describes the book production methods.

Table 6. Book production methods

Method	Description
`addManagedElement(managedElement)`	Append an IDMLManagedElement to the current operation in the current operationSet. It is an error to call this method if there is no current operation. The method returns a reference to the IDMLBook.
`addManagedElements(managedElements)`	Append a list of IDMLManagedElements to the current operation in the current operationSet. It is an error to call this method if there is no current operation. The method returns a reference to the IDMLBook.
`addRelationship(relationship)`	Append an IDMLRelationship to the current operation in the current operationSet. It is an error to call this method if there is no current operation. The method returns a reference to the IDMLBook.
`addRelationships(relationships)`	Append a list of IDMLRelationships to the current operation in the current operationSet. It is an error to call this method if there is no current operation. The method returns a reference to the IDMLBook.
`closeBook()`	Complete the book and close the file. It is an error to call this method unless the book has previously been opened using the `openBook()` method. It is also an error to call this method if the last call to `openOperationSet()` was not followed by a call to `closeOperationSet()`. The method returns a reference to the IDMLBook.
`closeOperation()`	Complete the current operation, as specified by the most recent call to the `openCreateOperation()`, `openDeleteOperation()`, or `openModifyOperation()` method. Following this call, there is no current operation defined until a subsequent call to open an operation. It is an error to call this method if there is no open operation. The method returns a reference to the IDMLBook.
`closeOperationSet()`	Complete the current operationSet. Following this call, there is no current operationSet defined until a subsequent openOperationSet call. It is an error to call this method if there is no open operationSet. The method returns a reference to the IDMLBook.

Table 6. Book production methods (continued)	
Method	**Description**
`create(source, timestamp, modelSchemaURI, modelSchemaVersion)`	This is a class scope method that creates an instance of an IDMLBook. The method accepts the following parameters: **source** The cdmManagementSoftwareSystem instance that identifies the source of discovery data in the book. **timestamp** The UTC time when the book was created. **modelSchemaURI** The URI for the schema to be used to validate the instances of managed elements and relationships in the operations.
`create(source, timestamp)`	This is a class scope method that creates an instance of an IDMLBook. Similar to `create(source, timestamp, modelSchemaURI, modelSchemaVersion)`, this method uses the base Common Data Model schema for validation instead of the user-specified `modelSchemaURI` and `modelSchemaVersion`.
`getBookName()`	Retrieve a character string with a name that conforms to the conventions for IdML book file names, based on the type, source, and timestamp properties.
`getSource()`	Retrieve the value of the source property.
`getTimestamp()`	Retrieve the value of the timestamp property.
`openBook(outputStream)`	Retrieve a reference to the IDMLBook which serves as the output stream to which the contents will be written. You can only call this method once for a particular IDMLBook.
`openCreateOperation(timestamp)`	Begin a create operation. You can specify a timestamp value for the operation. It is an error to call this method if there is no current operationSet. The method returns a reference to the IDMLBook.
`openDeleteOperation(timestamp)`	Begin a delete operation. You can specify a timestamp value for the operation. It is an error to call this method if there is no current operationSet. The method returns a reference to the IDMLBook.
`openModifyOperation(timestamp)`	Begin a modify operation. You can specify a timestamp value for the operation. It is an error to call this method if there is no current operationSet. The method returns a reference to the IDMLBook.

Table 6. Book production methods (continued)	
Method	**Description**
`openOperationSet(transactional)`	Open a new operationSet. The transactional parameter specifies whether the reader of the book should treat all operations in the operationSet as one transaction. You must call the `openBook()` method before calling `openOperationSet`. It is also an error to call the openOperationSet() mehtod if there is an existing operationSet open. The method returns a reference to the IDMLBook.
`openRefreshOperation(timestamp)`	Begin a refresh operation. You can specify a timestamp value for the operation. It is an error to call this method if there is no current operationSet. The method returns a reference to the IDMLBook.

Managed element properties and methods

You can use the properties and methods described in this section to create managed elements and add and retrieve associated attributes.

Properties

Table 7 on page 15 describes the managed element properties for the DLA book production API.

Table 7. Managed element properties	
Property	**Description**
`type`	The class type of the managed element.
`id`	A string that uniquely identifies the managed element within the IDMLBook.
`attributes`	A list of scalar attributes of the managed element. Each item in the list must be of type IDMLAttribute.

Methods

Table 8 on page 15 describes the managed element methods.

Table 8. Managed element methods	
Method	**Description**
`addAttribute(attribute)`	Add an attribute to the IDMLManagedElement attribute list. The method returns a reference to the IDMLManagedElement.
`addAttribute(name, value)`	Add an attribute with the specified name and value to the IDMLManagedElement attribute list. The method returns a reference to the IDMLManagedElement.

Table 8. Managed element methods (continued)	
Method	**Description**
`create(type, id)`	This is a class scope method that creates an instance of the IDMLManagedElement class using the following parameters: **type** The class of the managed element. It is the responsibility of the caller to ensure that this is a defined class in the model schema of the IDMLBook to which the managed element will be written. **id** Uniquely identifies the managed element within the IDMLBook. The source and target properties of the IDMLRelationship object refers to this value. The identifier does not need to be globally unique since it is used only within the IDMLBook. While the size of the id string is not bounded, you should use small strings to identify managed elements within an IDMLBook.
`create(type, id, attributes)`	This is a class scope method that accepts a list of attributes as a parameter and creates an instance of the IDMLManagedElement class. The list is copied into the IDMLManagedElement instance attribute list.
`getAttributes()`	Retrieve the list of attributes of the IDMLManagedElement.
`getId()`	Retrieve the ID property of the IDMLManagedElement.
`getType()`	Retrieve the type property of the IDMLManagedElement.

Attribute properties and methods

You can use the properties and methods described in this section to create attributes associated with managed elements.

Properties

Table 9 on page 16 describes the attribute properties for the DLA production API.

Table 9. Attribute properties	
Property	**Description**
`name`	The attribute name.
`value`	The attribute value.

Methods

Table 10 on page 17 describes the attribute methods.

Table 10. Attribute methods

Method	Description
`create(name, type, value)`	This is a class scope method that creates an instance of IDMLAttribute using the following parameters: **name** The name of the attribute. It is the responsibility of the caller to ensure that the name is valid for the IDMLManagedElement to which the IDMLAttribute is being added. **type** The type of the attribute. **value** The value of the attribute.
`getName()`	Retrieve the name property.
`getValue()`	Retrieve the value property.

Relationship properties and methods

You can use the properties and methods described in this section to create relationships between managed elements.

Properties

Table 11 on page 17 describes the relationship properties for the DLA production API.

Table 11. Relationship properties

Property	Description
`type`	The relationship type.
`source`	The identifier of the source IDMLManagedElement of the relationship.
`target`	The identifier of the target IDMLManagedElement of the relationship.

Methods

Table 12 on page 18 describes the relationship methods.

Table 12. Relationship methods

Method	Description
`create(type, source, target)`	This is a class scope method that creates an instance of IDMLRelationship by using the following parameters: **type** The type of the relationship. It is the responsibility of the caller to ensure that this value is a defined relationship type in the model schema of the IDMLBook to which the relationship is being written. **source** The ID property of the IDMLManagedElement that is the source of the relationship. It is the responsibility of the caller to ensure that a relationship is written to the IDMLBook after both the source and target IDMLManagedElements have been written. **target** The ID property of the IDMLManagedElement that is the target of the relationship. It is the responsibility of the caller to ensure that a relationship is written to the IDMLBook after both the source and target IDMLManagedElements have been written.
`create(type, source, target)`	This is a class scope method that creates an instance of IDMLRelationship. This method is similar to create(type, source, target) except that the source and target parameters are specified as strings instead of identifiers.
`getSource()`	Retrieve the property containing the source identifier.
`getTarget()`	Retrieve the property containing the target identifier.
`getType()`	Retrieve the type property.

Common Data Model helper methods

The idml_schema_2.4.jar file provides additional helper methods for working with metadata specific to the Common Data Model.

You can use the application programming interfaces in this .jar file in the creation of a Discovery Library Adapter; these methods simplify integration and use of the Common Data Model.

To use these methods, use the Common Data Model helper class setter methods for the metadata you want to work with. For example, you might use the addComputerSystem(ComputerSystem) helper method instead of the generic addManagedElement(managedElement) method.

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.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.