caAdapter Model Mapping Service (MMS) 4.1.1

User's Guide

Center for Biomedical Informatics and Information

September 30, 2009

CONTENTS

About This Guide

This section introduces you to the *caAdapter MMS 4.1.1 User's Guide*.

Topics in this section:

Purpose

This guide is the companion documentation to the caAdapter Model Mapping Service (MMS) tool. It includes information and instructions for using the caAdapter MMS tool graphical user interface (GUI) to map an object model to a data model. The caAdapter MMS tool is part of the caAdapter toolset which is an open source tool set that provides model mapping services and facilitates data mapping and transformation services.

Audience

Typical User

This guide is designed for users who wish to map a data model to an object modeling support of building caCORE-like applications.

Prerequisites

This guide assumes that you are familiar with the following concepts:

- UML modeling
- Enterprise Architect, or ArgoUML
- Object and data model terms and processes

Organization of This Guide

The *caAdapter MMS 4.1.1 User’s Guide* includes the following chapters:

- *Chapter 1, Overview of caAdapter,* on page 5 discusses the caAdapter architecture and related data standards.
- *Chapter 2, Using the caAdapter MMS Tool,* on page 13 provides detailed instructions for using the caAdapter GUI for object model to data model mapping.
- *Appendix A, References*, on page 43 provides a list of references used to produce this guide or referred to within the text.
- *caAdapter Glossary* on page 45 defines acronyms, objects, tools and other terms related to the caAdapter MMS Tool.

Recommended Reading

The following table lists resources that can help you become more familiar with concepts discussed in this guide.

Resource	***URL***
Unified Modeling Language (UML)	http://www.cdisc.org/models/sds/v3.1/

Click the hyperlinks throughout this guide to access more detail on a subject or product.

Text Conventions Used

This section explains conventions used in this guide. The various typefaces represent interface components, keyboard shortcuts, tool bar buttons, dialog box options, and text that you type.

Convention	*Description*	*Example*
Bold	Highlights names of option buttons, check boxes, drop-down menus, menu commands, command buttons, or icons.	Click **Search**.
URL	Indicates a Web address.	http://domain.com
text in SMALL CAPS	Indicates a keyboard shortcut.	Press ENTER.
text in SMALL CAPS + text in SMALL CAPS	Indicates keys that are pressed simultaneously.	Press SHIFT + CTRL.
Italics	Highlights references to other documents, sections, figures, and tables.	See *Figure 4.5*.
`Italic boldface monospaced type`	Represents text that you type.	In the **New Subset** text box, enter ***`Proprietary Proteins.`***
Note:	Highlights information of particular importance	**Note:** This concept is used throughout the document.
{ }	Surrounds replaceable items.	Replace {last name, first name} with the Principal Investigator's name.

Credits and Resources

The following people contributed to the development of this document.

caAdapter Development and Management Teams		
Development	***User's Guide***	***Program Management***
Ki Sung Um[3]	Ki Sung Um[3]	Anand Basu [1]
Eugene Wang[3]	Eugene Wang[3]	Christo Andonyadis [1]
Ye Wu [3]	Ye Wu[3]	Sichen Liu [1]
Wendy Ver Hoef [2]	Wendy Ver Hoef [2]	Sharon Settnek [3]
	Carolyn Kelley Klinger [4]	Smita Hastak [2]
[1] National Cancer Institute Center for Bioinformatics and Information Technology (NCI CBIIT)		[2] ScenPro, Inc.

caAdapter Development and Management Teams		
Development	***User’s Guide***	***Program Management***
[3] Science Application International Corporation (SAIC)		[4] Lockheed Martin Management System Designers

Contacts and Support	
Application Support	http://ncicb.nci.nih.gov/NCICB/support Telephone: 301-451-4384 Toll free: 888-478-4423

LISTSERV Facilities Pertinent to caAdapter		
LISTSERV	***URL***	***Name***
caAdapter_Users	https://list.nih.gov/archives/caadapter_users-l.html	caAdapter Users Discussion Forum
caBIO_Users	https://list.nih.gov/archives/cabio_users.html	caBIO Users Discussion Forum
caBIO_Developers	https://list.nih.gov/archives/cabio_developers.html	caBIO Developers Discussion Forum

Release Schedule
This guide has been updated for the caAdapter MMS 4.1.1 release. It may be updated between releases if errors or omissions are found.

CHAPTER

1

OVERVIEW OF CAADAPTER

This chapter provides an overview of caAdapter, its architecture, its various tools, and its related data standards.

Topics in this chapter include:

About caAdapter

The caAdapter (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caadapter) is an open source tool set that provide model mapping services in support of building caCORE-like applications and facilitates data mapping and transformation among different kinds of data source including HL7 v2 messages, HL7 v3 messages, and Study Data Tabulation Model (SDTM) data sets. caAdapter has a component-based architecture with two major architectural components, the Core Engine and the Mapping Tool.

The Core Engine includes model mapping service (MMS), data mapping and transformation service (MTS), validation component, and other sub-components. The MMS component maps object model to data model. The MTS component facilitates data mapping and transformation among different kinds of data sources that support data sharing at NCI CBIIT (http://ncicb.nci.nih.gov) and/or cancer centers as part of the cancer Biomedical Informatics Grid (caBIG) (http://caBIG.nci.nih.gov) solution. The validation component validates message with vocabulary and HL7 structural attributes.

The caAdapter MMS component allows users map an object to a data model using drag-and-drop capabilities. It parse and load data and object models from an xmi file, add caCORE SDK-required tags and tag values into the xmi file, and generate Hibernate mapping files.

caAdapter integrates with NCI CBIIT cancer Common Ontologic Representation Environment (caCORE) components (http://ncicb.nci.nih.gov/NCICB/infrastructure). See the caCORE Technical Guide (ftp://ftp1.nci.nih.gov/pub/cacore) and the caCORE Software Development Kit Programmer's Guide (ftp://ftp1.nci.nih.gov/pub/cacore/SDK) for more information.

caAdapter Core Engine Architecture

Figure 1.1 illustrates the caAdapter core engine architecture design including its subsystems and components.

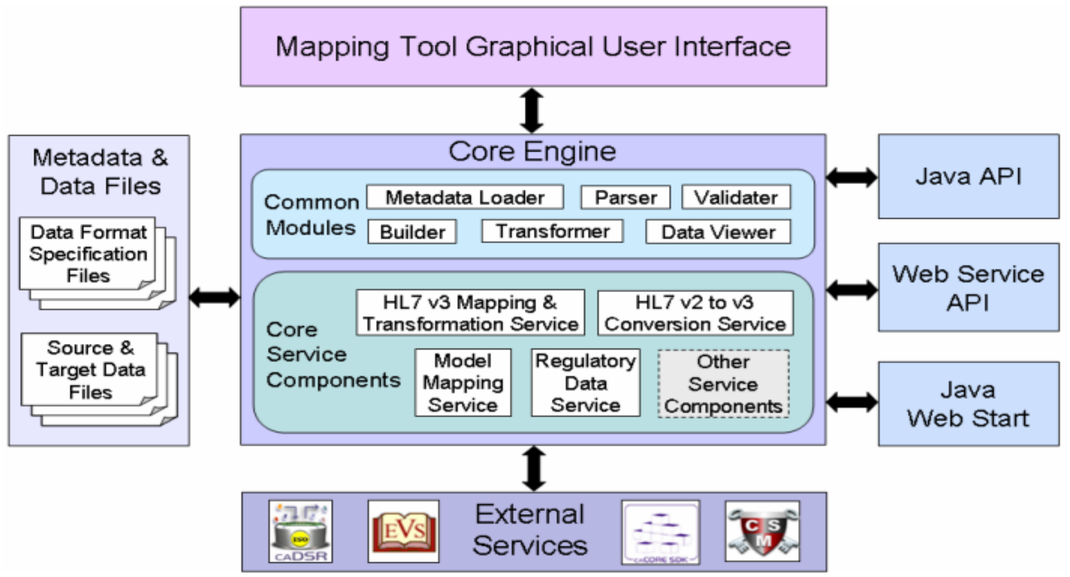


Figure 1.1 caAdapter Core Engine Architecture

The main features of the caAdapter core engine are

- Metadata Loader - loads metadata information for HL7 v2 and v3, CSV, and XMI files
- Parser - parses HL7 v2 and v3 messages, CSV, and XMI files
- Builder - builds HL7 v3 messages and SDTM datasets
- Transformer - transforms source to target data based on mapping specifications
- Data Viewer - displays transformed data in the UI
- Validater - validates schema files and mapping specifications

caAdapter Mapping Tool

The caAdapter Mapping Tool is a graphical application for mapping elements between data structures elements or model structures elements.

The mapping tool provides the following:

- Source and Target Specification - graphical interface for defining input and output formats.
- User Interface - simple mechanism for mapping source to target elements containing tree structure, drag-and-drop functionality, and functions and property definitions.
- Mapping Functions - capability to do simple source data manipulation.
- Transformation Service - generation of HL7 v3 XML message instances and SDTM text files from a source database based on user-defined mapping specifications.
- Validation - capability to validate structure and mapping specifications.

About HL7

Health Level Seven (HL7) (http://www.hl7.org/) is one of several American National Standards Institute (ANSI)-accredited Standards Developing Organizations (SDOs) operating in the health care arena. HL7 provides standards for data exchange to allow interoperability between health care information systems. It focuses on the clinical and administrative data domains. The standards for these domains are built by consensus by volunteers—providers, payers, vendors, government—who are members in the not-for-profit HL7 organization.

HL7 version 2 (v2) is a messaging standard that focuses on syntactic data interchange. HL7 messaging (v2 or higher) has been recommended as a data exchange standard by the e-Government initiative. In fact, various releases of this version are in use in over 90% of U.S. hospitals, and v2 is considered the most widely implemented standard for healthcare information in the world. However, since it lacks an explicit methodology, conformance rules, and grouping of messages, it cannot be considered an interoperability standard.

HL7 v2 messages are composed of segments (individual lines in a message) which are composed of fields (data values) which may in turn be composed of components and sub-components. Several different delimiters or field separators are used to mark boundaries between the various elements. Specifications for messages using these structures are published in a text document format which does not easily lend itself to being computable. Furthermore, messages are often customized at local sites making it difficult to share messages between sites.

HL7 version 3 (v3) is a messaging standard aimed to address some of the problems of HL7 v2 standard. The HL7 v3 standard supports the key goal of the HL7 community: syntactic and semantic interoperability. It achieves this goal by what are commonly called the four pillars of semantic interoperability:

1. A common Reference Information Model (RIM) spanning the entire clinical, administrative, and financial healthcare universe. The RIM is the cornerstone of the HL7 v3 development process. An object model created as part of the v3 methodology, the RIM is a large pictorial representation of the clinical data domains and identifies the life cycle of events that a message or groups of related messages will carry. It is a shared model between all the domains and is the model from which all domains create their messages. Explicitly representing the connections that exist between the information carried in the fields of HL7 messages, the RIM is essential to HL7's ongoing mission of increasing precision and reducing implementation costs.
2. A well-defined and tool-supported process for deriving data exchange specifications from the RIM. HL7 has defined a methodology and process for developing specifications, artifacts to document the models and specifications, tools to generate the artifacts and an organization for governing the overall process of standards development. Such structure avoids ambiguity common to many existing standards.
3. A formal and robust data type specification upon which to ground the RIM. Data types are the basic building blocks of attributes. They define the structural format of the data carried in the attribute and influence the set of allowable values an attribute may assume. HL7 defines an extensive set of complex data types which provide the structure and semantics needed to describe data in the healthcare arena.
4. A formal methodology for binding concept-based terminologies to RIM attributes. Within HL7, a vocabulary domain is the set of all concepts that can be taken as valid values in an instance of a coded field or attribute. HL7 has defined vocabulary domains for some attributes to support use of the RIM in messages. It also provides the ability to use, document, and translate externally coded vocabularies in HL7 messages.

The specifications that are developed upon this foundation are documented in a progressive set of artifacts that represent varying levels of abstraction of the domain data. The artifacts go from purely abstract and universal in scope to implementation-specific and very narrow in subject matter:

- The RIM is the foundational Unified Modeling Language (UML) class diagram representing the universe of all healthcare data that may be exchanged between systems.
- A Domain Message Information Model (DMIM) is a subset of the RIM that includes RIM class clones, attributes, and associations that can be used to create messages for a particular domain (a particular area of interest in healthcare). DMIMs use HL7 modeling notation, terminology, and conventions.
- A Refined Message Information Model (RMIM) is a subset of a DMIM that is used to express the information content for an individual message or set of messages with annotations and refinements that are message specific.
- A Model Interchange Format (MIF) is an XML representation of the information contained in an HL7 specification, and is the format that all HL7 v3 specification authoring and manipulation tools will be expected to use.

- A Message Type (MT) is the specification of an individual message in a specific business domain.

While the HL7 standard is not implementation specific, the caAdapter implements the HL7 mapping and transformation features on the basis of the MIF and MT artifacts. The MIF and MT promote caAdapter uses XML as its implementation technology.

The NCI CBIIT provides training resources to assist the caBIG community and other interested parties in implementing HL7 v3 messaging. These resources include online tutorials, self-paced training, and links to HL7 resources (http://ncicb.nci.nih.gov/infrastructure/cacore_overview/caadapter/indexContent/HL7_Tutorial).

About the Study Data Tabulation Model (SDTM)

The Study Data Tabulation Model, or SDTM, is a set of standards developed by the Clinical Data Interchange Standards Consortium (CDISC). It provides structured guidelines for submitting study data tabulations to a regulatory authority such as the United States Food and Drug Administration (FDA).

SDTM datasets are organized by *domains*, where each domain contains a list of *variables*. Each domain is identified by a two-letter acronym. The variables within a domain is referred with an eight-character naming convention. An example domain is *Demographics*, which is referred to by the acronym *DM*. The Demographics dataset contains variables such as patient name, patient date of birth, race, and sex.

Domains are grouped into *classes*. Domain classes include the following:

- Trial Design
- Interventions
- Events
- Findings, and,
- Special Purpose

Figure 1.2 lists SDTM Domain Classes and associated Domains.

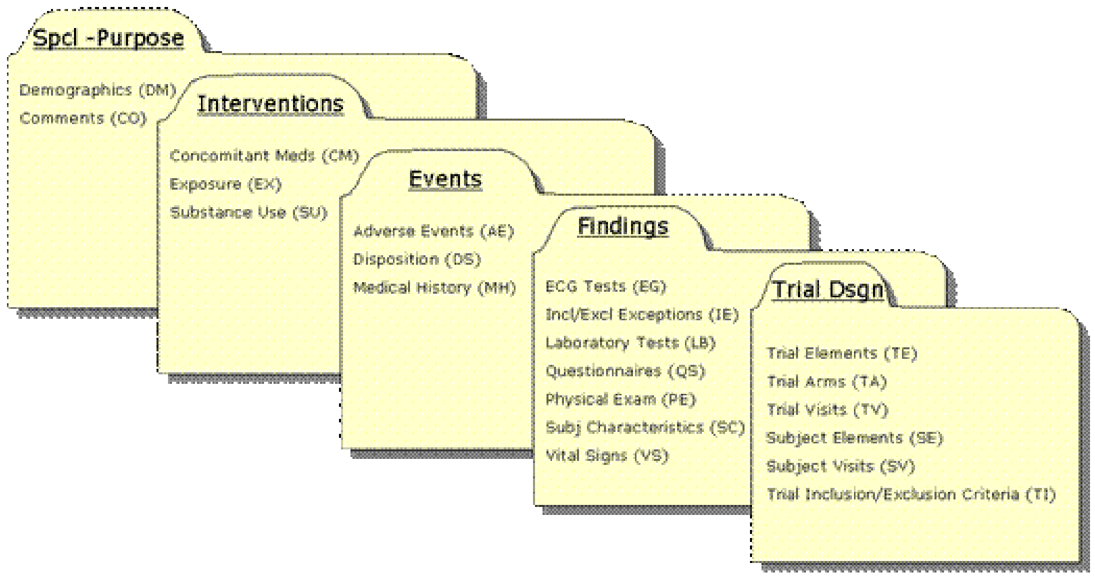


Figure 1.2 SDTM Domain Classes

SDTM dataset structures are fully defined in the guide "Study Data Tabulation Model Implementation Guide: Human Trials," which is available from the CDISD web site at (http://www.cdisc.org). Furthermore, SDTM datasets are defined in an XML document often referred to as the `define.xml`. CDISC provides a sample `define.xml` document which was used in the implementation of the CSV to SDTM Mapping and Transformation component of caAdapter.

About the Object and Data Model

The caAdapter Model Mapping Service takes advantage of caAdapter's mapping infrastructure to facilitate object to data model mapping. The model mapping service requires an `.xmi` file (with full Enterprise Architect (EA), or ArgoUML, round-trip capability) that includes a data and an object model. It loads both models into the tool and gives the user the ability to map object elements to a table elements using drag-and-drop capability. Once the mapping is completed, caAdapter adds all caCORE SDK-required tagged values into the `.xmi` file. This module also has the capability to create Hibernate object-relational mapping files.

This guide focuses on using this feature of caAdapter.

Prerequisites for Using the caAdapter Mapping Tool

Successful use of the caAdapter mapping tool requires the following prerequisites:

- Familiarity with UML modeling of object and data models
- Training on the caAdapter Mapping Tool
- Familiarity with this document

Resources for Installing caAdapter MMS Tool

Complete instructions for installing caAdapter MMS are located in the *caAdapter MMS Installation Guide* at http://gforge.nci.nih.gov/docman/?group_id=77.

Starting the caAdapter MMS Tool

Starting the caAdapter MMS Tool from the Binary Distribution

To launch the caAdapter MMS Tool GUI, follow these steps:

1. In a Command Prompt window, enter `cd {home directory}` to go to your home directory (Windows example: `C:\caadapter`).
2. Enter `java -jar caadapter_ui.jar`.

   The Welcome to the caAdapter screen momentarily appears, followed by the caAdapter GUI.

Starting the caAdapter MMS Tool from the Source Distribution

To launch the caAdapter MMS Tool, follow these steps:

1. In a command prompt window, enter `cd {home directory}` to go to your caAdapter home directory (Windows example: `C:\caadapter`).
2. Enter `cd ..\caadaper`
3. Enter `ant`
4. Enter `run.bat`.

   The Welcome to the caAdapter screen momentarily appears, followed by the caAdapter Mapping Tool GUI.

Starting the caAdapter MMS Tool from the Windows Distribution

To launch the caAdapter MMS Tool, select **caAdapter** from the Start menu.

Starting the Mapping Tool on the Web (WebStart)

You can also use caAdapter MMS in your web browser by entering the following URL: http://caadapter.nci.nih.gov/caadapter-mms/caadapter-mms.jnlp.

CHAPTER 2
USING THE CAADAPTER MMS TOOL

This chapter describes how to use caAdapter MMS Tool to facilitate object to data model mapping.

Topics in this chapter include:

About the caAdapter Model Mapping Service

The caAdapter Model Mapping Service takes advantage of caAdapter's mapping infrastructure to facilitate object to data model mapping. It loads a UML model and presents graphically with tree structure of object model in left panel and data model in right panel. User is able to set up mapping from a tree node in left panel to a tree node in the right panel drag-and-drop capability. At each mapping step, the caAdapter mapping engine validates mapping rules. If a mapping is valid, it adds caCORE SDK-required tagged values into the UML model. If a mapping is invalid, it displays a error message. The caAdapter Model Mapping Service supports UML models created by two kind of UML modeling tools: Enterprise Architect (EA), or ArgoUML. It processes the UML models exported `.xmi` files by UML modeling tools.

Mapping Tool Operational Scenario for Model Mapping Service

The essential operational scenario is to generate caBIG silver-level compliant of caCORE-like software, using caCORE SDK tool sets. In this scenario, user first develops a UML model containing both an object model and a data model using EA, or ArgoUML. Once the UML model is complete, user needs add caCORE tags on the model defining dependence from objects to tables, associations from attributes of objects to columns of tables, associations between objects, and primary key features of tables. If all tags are added correctly, user will be able to export the UML as an `.xmi` file and forward it to caCORE SDK to generate a caCORE-like application.

It has been proved error prone and very time consuming to define all the caCORE tags manually. The caAdapter model mapping service component automates this process. The user can map setup all required mapping using drag-and-drop capability. For each valid mapping, all the caCORE SDK required tags are added automatically on the `.xmi` file. The user can then use the annotated `.xmi` file for caCORE SDK code generation.

The alternate operational scenario is for the users who are not using the caCORE SDK but want to map an object model to a data model. In this scenario, user can use the Model Mapping Service tool to perform the mapping, and create Hibernate object-relational mapping files for the UML model.

Using the caAdapter Model Mapping Service

The caAdapter Model Mapping Service provides the following features:

- Parse and graphically present the `.xmi` file of a UML model containing object model and data model
- Setup object-relational mapping between object model to data model using drag-and-drop capability. This includes dependence from objects to tables, associations from attributes of objects to columns of tables, associations between objects, and primary key features of tables.
- Add caCORE SDK required tags and tag values into the `.xmi` file
- Generate Hibernate object-relational mapping files

The process flow for integrating the caAdapter Model Mapping Service with other components follows these steps, assuming, for example, EA was used as the UML tool:

1. Develop a UML model with EA containing an object model and a data model.
2. Export the UML model as an `.xmi` file from EA so that the caAdapter Model Mapping Service can process the `.xmi` file.
3. Setup object-relational mapping by dragging and dropping.

As illustrated in *Figure 2.1*, this process enables caAdapter to annotate the original XMI file with caCORE SDK required tags to generate caCORE-like application. Alternatively, caAdapter can directly generate the Hibernate object-relational mapping files.

.

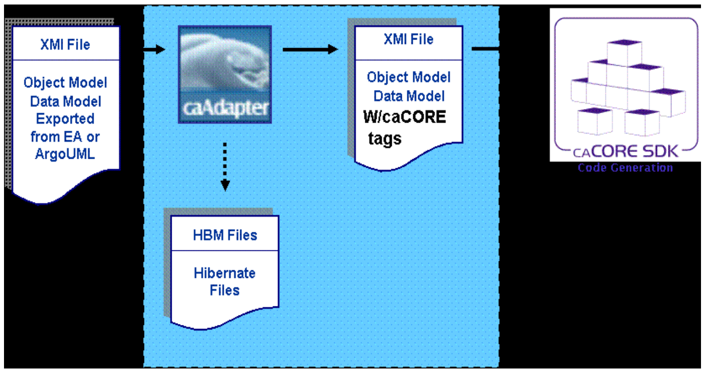

XMI File
Object Model Data Model Exported from EA or ArgoUML
caAdapter
XMI File
Object Model Data Model W/caCORE tags
caCORE SDK
Code Generation
HBM Files
Hibernate Files

Figure 2.1 caAdapter Model Mapping Service - Overall Process

The following subsections describe each of the mapping process steps in detail.

Generating an XMI File from EA

Perform the following steps to export a UML model as an `.xmi` fileit.

1. Open the `.eap` file (that is, the file that contains the object and data models).
2. In the Project View pane, right-click **Logical View**. A popup menu appears.
3. Select **Import/Export** > **Export package to XMI file** (*Figure 2.2*). The Export Package to XMI dialog box appears.

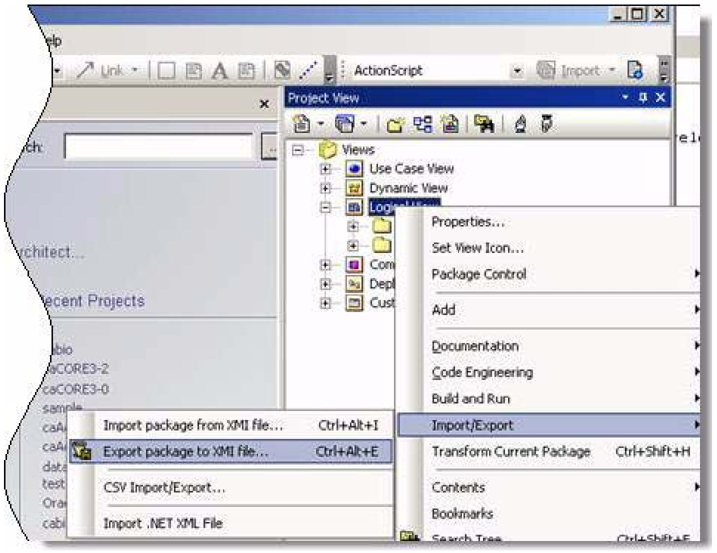
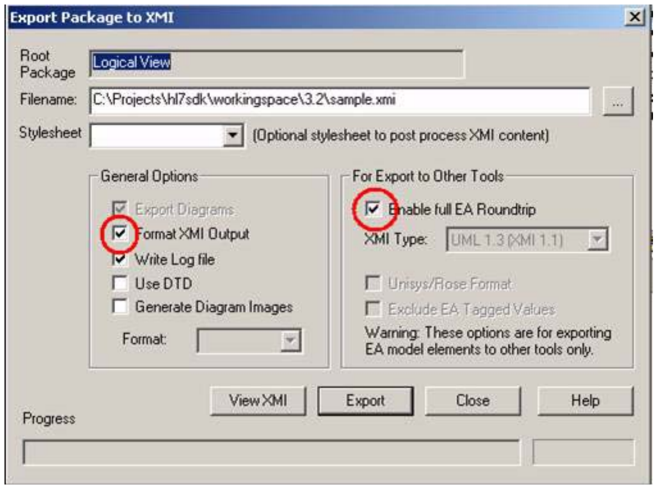


Figure 2.2 Exporting an XMI File from EA

4. In the **Filename** field, specify the output file name of the XMI file.
5. Check the following boxes:
   - **Format XMI Output**
   - **Enable Full EA Roundtrip**

6. Click **Export**.

   The generated XMI file can now be processed by the caAdapter Model Mapping Service module (*Figure 2.3*).



Export Package to XMI

Root Package: Logical View

Filename: C:\Projects\hl7sdk\workingspace\3.2\sample.xmi

Stylesheet: (Optional stylesheet to post process XMI content)

General Options
- ☑ Export Diagrams
- ☑ Format XMI Output
- ☑ Write Log file
- ☐ Use DTD
- ☐ Generate Diagram Images
- Format:

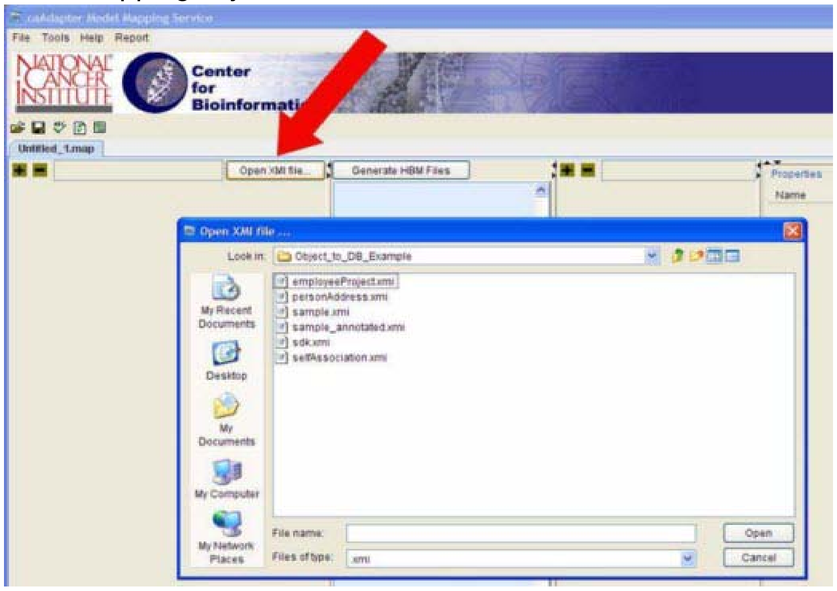
For Export to Other Tools
- ☑ Enable full EA Roundtrip
- XMI Type: UML 1.3 (XMI 1.1)
- ☐ Unisys/Rose Format
- ☐ Exclude EA Tagged Values
- Warning: These options are for exporting EA model elements to other tools only.

Figure 2.3 Options to Export XMI File from EA

Creating an Object Model to Data Model Map Specifications

Perform the following steps to create a new map specification.

1. Select **File** > **New** > **Model Mapping Service** > **Object Model to Data Model Map Specification** (*Figure 2.4*) to open a new mapping tab with empty source and destination panels.
2. Click **Open XMI file…** to display the Open XMI file … dialog box (*Figure 2.4*).
3. Select the XMI file. After the XMI file is loaded, caAdapter displays the object model in the left panel and the data model in the right panel.
4. Start mapping objects and attributes to tables and columns.



Open XMI file ...

Look in: Object_to_DB_Example

- employeeProject.xmi
- personAddress.xmi
- sample.xmi
- sample_annotated.xmi
- sdk.xmi
- selfAssociation.xmi

File name:

Files of type: xmi

Figure 2.4 Open XMI File button

Opening an Existing Object to Data Model Mapping Specification

Perform the following steps to open an existing map specification.

1. Select **File** > **Open** > **Object Model to Data Model Map Specification**. The Open Map File dialog box appears.
2. Select the XMI file and click **Open**.

Basic Mapping

Perform the following steps to create dependency, attribute, and association mappings from an object model to a data model.

1. Select a source tree node (object, attribute, or association end) from the object model and drag it to the appropriate target tree node (table, column or foreign key) in the data model. The cursor indicates whether the source tree node is allowed to be mapped to the target tree node (*2.*). Drop the source tree node on the target tree node.
2. Once a source tree node is mapped to a target tree node, a mapping line appears between them in the mapping panel. *Figure 2.5* shows a mapping line between `Amendment` in the object model, on the left, and `AMENDMENT` in the data model, on the right.

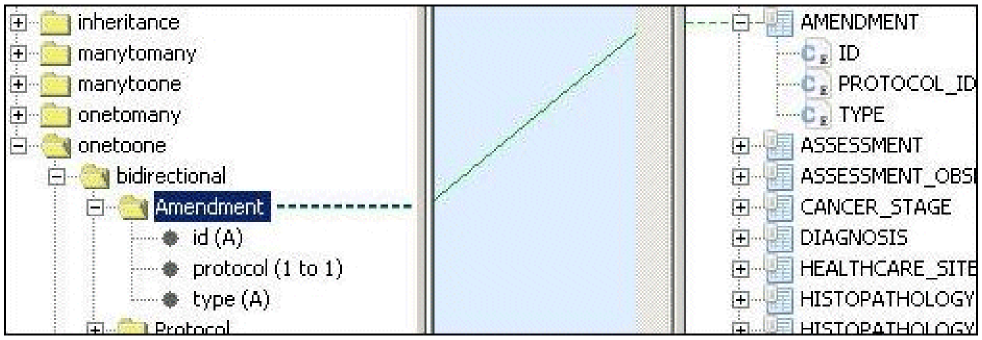


Figure 2.5 Mapping line between source tree node and target tree node

Dependency Mapping (Object to Table)

A dependency mapping defines the relationship between an object and its persistence table. Perform the following steps to create a dependency mapping.

1. Select a object tree node from the object model on the left panel. The example in *Figure 2.6* shows `HealthcareSite`.
2. Click and drag the `HealthcareSite` node to the `HEALTHCARE_SITE` node in the data model on the right panel.

   A mapping line between `HealthcareSite` in the object model and `HEALTHCARE_SITE` in the data model is now visible. Dependency mapping lines are green.

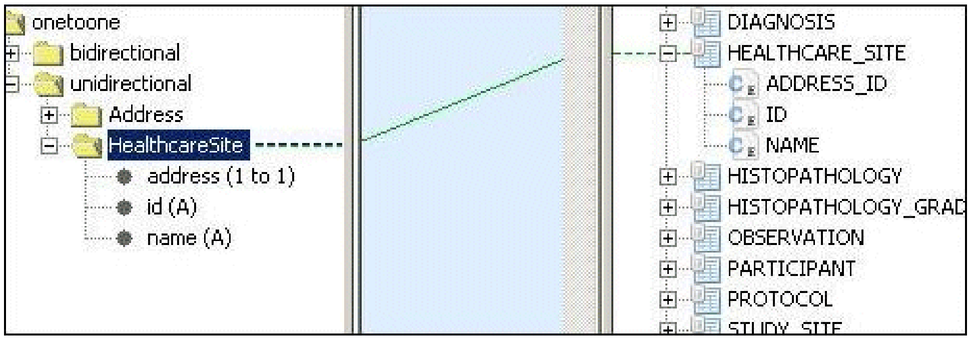


Figure 2.6 Dependency Mapping

Note: One object can only map to one table since it can only be persisted into one table; One table can only be mapped from multiple objects to support the One Table Per Hierarchy inheritance mapping strategy (see *Table Per Class Hierarchy* on page 26).

Attribute Mapping

An attribute mapping defines the relationship between an attribute in the object and a column in the persistence table of this object. Perform the following steps to set up an attribute mapping.

1. Select '`id (A)`' in the object model and drag it to ID in the data model.

   Note: The example in *Figure 2.7* shows the attribute `id (A)` for the class `HealthcareSite`.

   A mapping line should be visible between the attribute and column. Attribute mapping lines are color-coded blue.

2. Repeat this for '`name (A)`' to `NAME`.

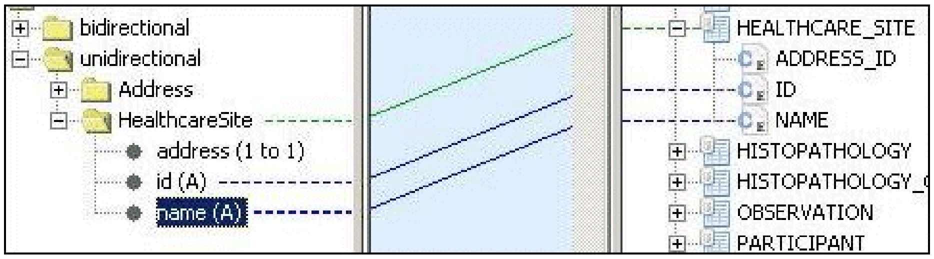
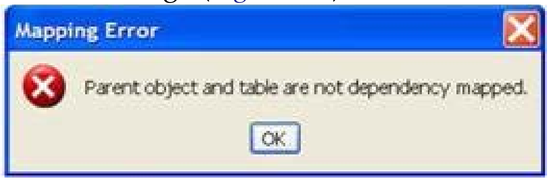


Figure 2.7 Attribute Mapping

Note: If the object has not already been mapped to the table, attempting to map an object's attributes to the table's columns will result in an error message (*Figure 2.8*).


Mapping Error

Parent object and table are not dependency mapped.

Figure 2.8 Attribute Mapping error message

Note: Before any attribute mapping can be performed, the user has to complete dependency mapping first.

Association Mapping

An association mapping defines the relationship between one end of an association listed under an object in the object model and a foreign key column in a table in the data model. Perform the following steps to setup an association mapping.

1. First create the dependency mapping between the object and the table. For example, in *Figure 2.9*, the green line shows a dependency between '`HealthcareSite`' and '`HEALTHCARE_SITE`'.

   There exists an one-to-one association from '`HealthcareSite`' object and '`Address`' object in the object model.

2. Click and drag '`address (1 to 1)`' association end under '`HealthcareSite`' object in the object model panel to `ADDRESS_ID` column of the `HEALTHCARE_SITE` table. When complete, the final result should look like *Figure 2.9*. Association mapping lines are color-coded red.

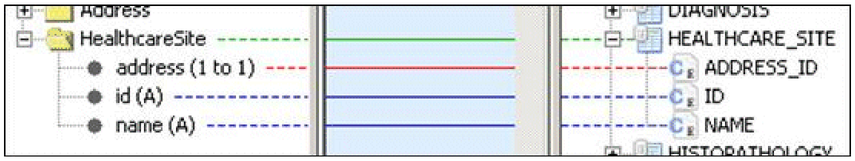


Figure 2.9 Association Mapping

Deleting Mapping Lines

Perform the following steps to delete a mapping line.

1. Select the mapping line by left clicking it in the mapping panel. The line is highlighted.

2. Right-click the highlighted mapping line and select **Delete** (*Figure 2.10*). The line is removed from the mapping panel.

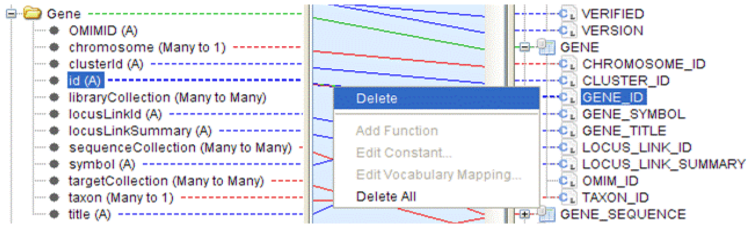


Figure 2.10 Deleting Mapping Lines

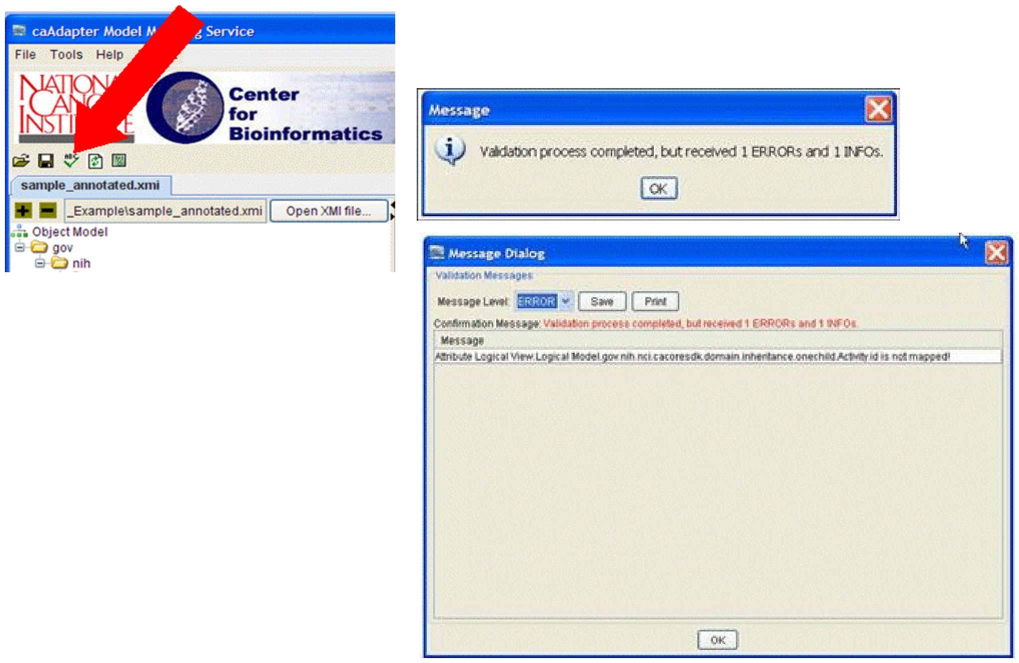
Note: A dependency mapping can only be deleted if no child node is mapped.

Validating Mapping Specifications

Validating a mapping specification identifies any pertinent business rules that have been violated and indicates any changes that need to be made. Perform the following steps to validate the object to data model mapping specification.

- Click the **Validate** button (top of *Figure 2.11*). The following message displays: `Validation process completed successfully with no message received`.

  If there are errors in the validation process, the following message displays (see *Figure 2.11*): `Validation process completed but received <some number> ERRORs`. Error messages may identify what actions to perform to correct errors, while warnings and informational messages may require no changes at all. It is recommended that mappings be re-validated after changes are made.



Validation process completed, but received 1 ERRORs and 1 INFOs.

Confirmation Message: Validation process completed, but received 1 ERRORs and 1 INFOs.

Attribute Logical View.Logical Model.gov.nih.nci.cacoresdk.domain.inheritance.onechild.Activity.id is not mapped!

Figure 2.11 Validate Mapping Specification

Saving Mapping Specifications

To save a mapping specification, select **File** > **Save**. caAdapter saves the XMI file annotated with all of the tags required by caCORE. The Save Complete dialog box appears.

You can use the annotated XMI file to generate a caCORE-like application.

Generating Hibernate Object-relational Mapping Files

An alternative to creating caCORE SDK APIs is to generate Hibernate files and use those files in an application to access data from a database. Perform the following steps to generate Hibernate object-relational mapping files from the current object to data model mapping.

1. Click the **Generate HBM Files** button. The Open dialog box appears (*Figure 2.12*).
2. Select a directory in which to save the HBM file(s) and click **Open**. The HBM object-relational mapping files are saved to that directory.

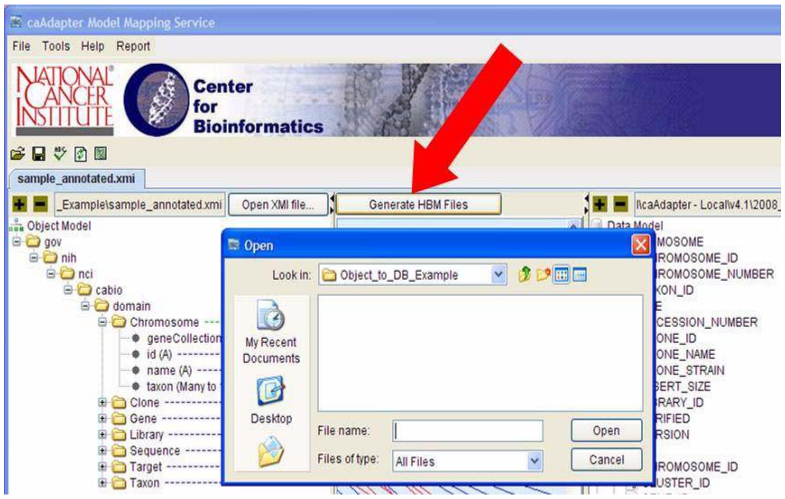


Figure 2.12 Generate HBM Files

The Seven Association Mapping Scenarios

Before performing any of the following mapping scenarios, all dependency mappings between objects and tables have to be completed.

One-to-One Bi-Directional

The following mapping rules apply to one-to-one bi-directional association:

- Both association end objects are mapped to their persistence tables.
- Both the association ends are navigable or visible in the left mapping panel.
- Maybe only persistence table for association end objects has the foreign key column of the persistence table for the other association end object.
- Map any association end to the foreign key column of its persistence table.
- Only one association end need to be mapped; the other association end does not need to be mapped

In the example (Protocol and Amendment) in *Figure 2.13*, drag the association end node (`Amendment.protocol (1 to 1)`) and drop it onto the foreign key column (`PROTOCOL_ID`) of the persistence table (`AMENDMENT`).

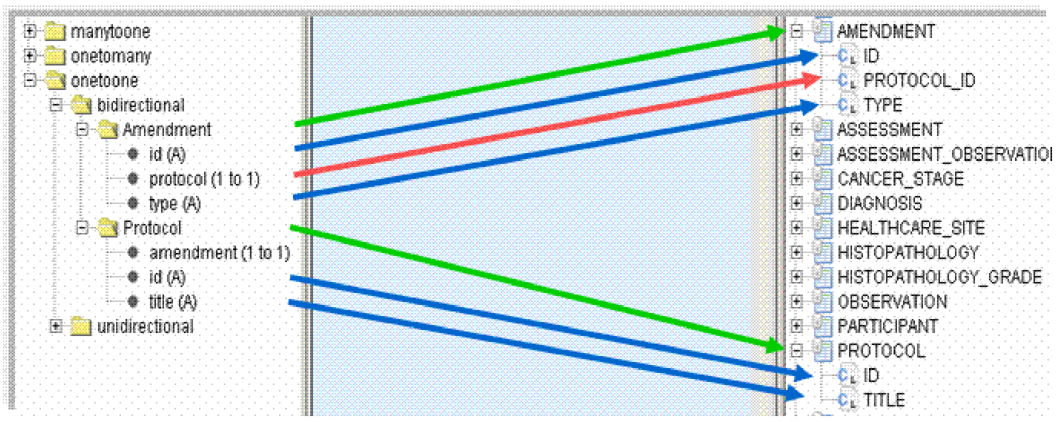


Figure 2.13 One-to-One Bi-Directional Mapping

One-to-One Uni-Directional

The following mapping rules apply to one-to-one uni-directional association:

- Both association end objects are mapped to their persistence tables.
- Only one association end (source association end) is navigable or visible in the left mapping panel.
- The persistence table of the source association table must have the foreign key column of the persistence table for the invisible target association end object.
- Map the visible association end to the foreign key column of its persistence table.

In the example (`HealthcareSite` and `Address`) in *Figure 2.14*, drag the association end (`HealthcareSite.address(1 to 1)`) and drop it onto the foreign key (`ADDRESS_ID`) of its persistence table (`HEALTHCARE_SITE`).

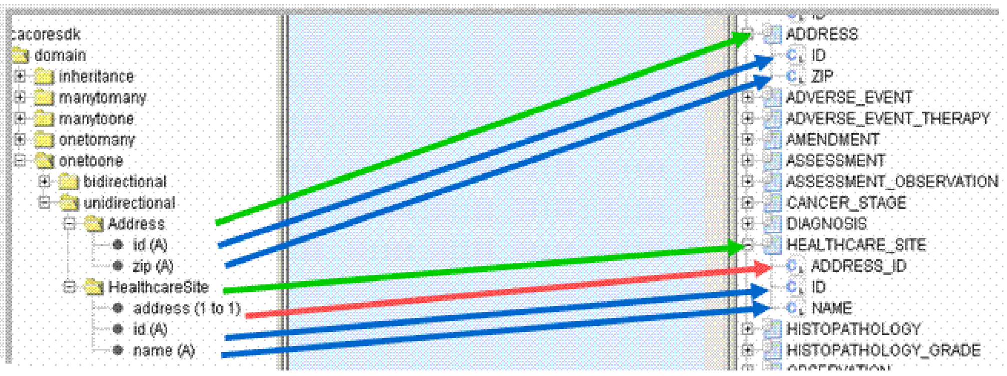


Figure 2.14 One-to-One Uni-Directional Mapping

One-to-Many/Many-to-One Bi-Directional

The following mapping rules apply to one-to-many or many-to-many bi-directional association:

- Both association end objects are mapped to their persistence tables.

- Both the association ends are navigable or visible in the left mapping panel.
- The persistence table of the “many” side association end object must have the foreign key column of the persistence table for the other association end object.
- Map the “many” side association end to the foreign key column of its persistence table.
- Only the “many” side association end need to be mapped; the other “one” association end does not need to be mapped.

In the example (`AdverseEvent` and `AdverseEventTherapy`) in *Figure 2.15*, drag the association (`AdverseEventTherapy.adverseEvent (Many to 1)`) and drop it onto the foreign key (`ADVERSE_EVENT_ID`) of the corresponding table (`ADVERSE_EVENT_THERAPY`).

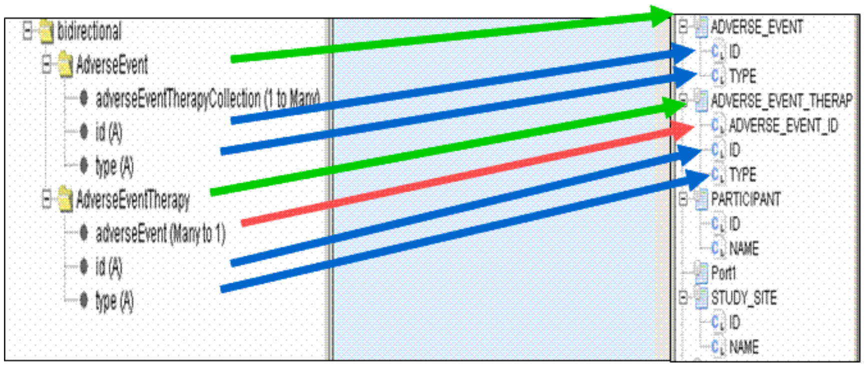


Figure 2.15 One-to-Many Bi-Directional Mapping

Many-to-One Uni-Directional

The following mapping rules apply to many-to-one uni-directional association:

- Both association end objects are mapped to their persistence tables.
- Only the “many” side association end (source association end) is navigable or visible in the left mapping panel.
- The persistence table of the source association table must has the foreign key column of the persistence table for the invisible target association end object.
- Map the visible association end to the foreign key column of its persistence table.

In the example (`Histopathology` and `HistopathologyGrade`) in *Figure 2.16* , drag the association (`HistopathologyGrade.histopathology(Many to 1)`) and drop it onto the foreign key (`HISTOPATHOLOGY_ID`) of the corresponding table (`HISTOPATHOLOGY_GRADE`).

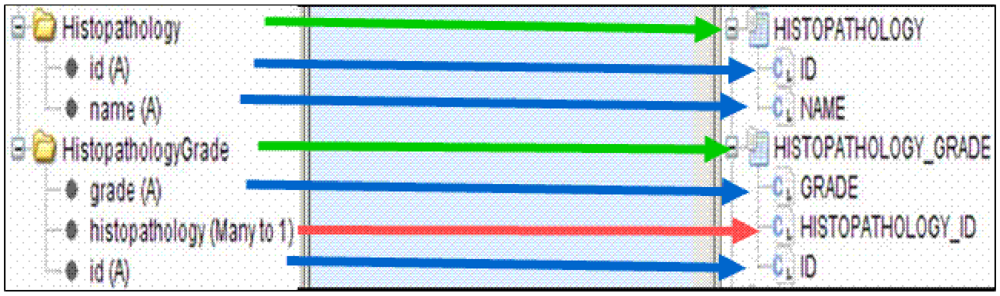


Figure 2.16 Many-to-One Uni-Directional Mapping

One-to-Many Uni-Directional

The following mapping rules apply to one-to-many uni-directional association:

- Both association end objects are mapped to their persistence tables.
- Only the "one" side association end (source association end) is navigable or visible in the left mapping panel.
- The persistence table of the source association end object does not have the foreign key column of the persistence table for the invisible target association end object.
- The persistence table of the invisible target association end has the foreign key column of the persistence table for the source association end object.
- Map the visible "one" side association end to the foreign key column of the persistence table of the invisible "many" side association end.

In the following example, there is an one-to-many association from `KeyChain` to `LatchKey`. `KeyChain` is mapped to `KEYCHAIN` table. `LatchKey` is mapped to `LATCH_KEY` table. `LATCH_KEY` has a foreign key column `KEYCHAIN_ID` referring to `KEYCHAIN` table. Drag the visible "one" side association end (`KeyChain.keyCollection (1 to many)`) and drop it onto the foreign key (`LATCH_KEY.KEYCHAIN_ID`) of the persistence table (`LATCH_KEY`) of object `LatchKey`.

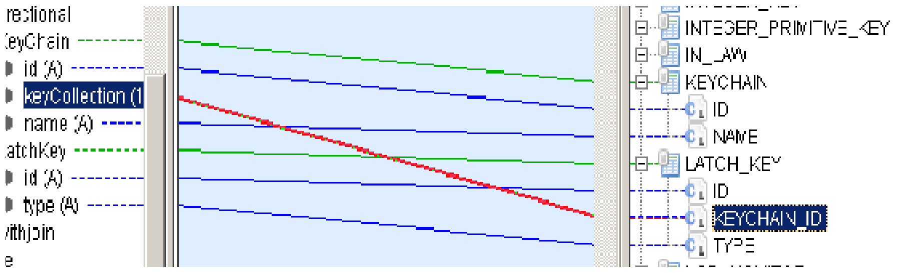


Figure 2.17 One-to-Many Uni-Directional Mapping

Many-to-Many Bi-Directional

The following mapping rules apply to many-to-many bi-directional association:

- Both association end objects are mapped to their persistence tables.

- Both the association ends are navigable or visible in the left mapping panel.
- A "correlation-table" is required to have the foreign key column of the persistence table for the both association end objects.
- Map both the association end to the foreign key column of the "correlation-table".

In the example in *Figure 2.18*, there is a many-to-many bi-directional association between `StudySite` and `Participant`. `STUDY_SITE_PARTICIPANT` is the intersection table (typically the name of the correlation table). Then, drag both ends of the associations and drop them onto the two corresponding columns in the correlation table.

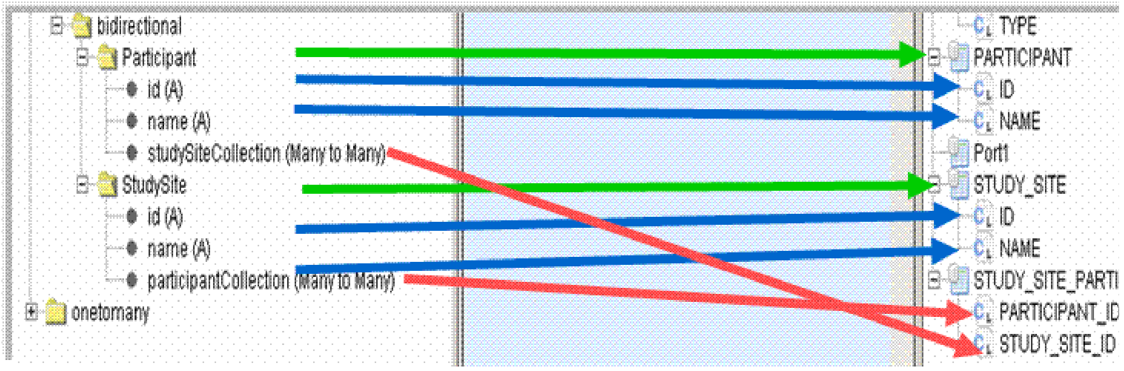


Figure 2.18 Many-to-Many Bi-Directional Mapping

Many-to-Many Uni-Directional

The following mapping rules apply to many-to-many uni-directional association:

- Both association end objects are mapped to their persistence tables.
- Only one association end is navigable or visible in the left mapping panel.
- A "correlation-table" is required to have the foreign key column of the persistence table for the both association end objects.
- Map only the visible the association end to the foreign key column of the "correlation-table".

In the example in *Figure 2.19*, there is a many-to-many uni-directional association between `Assessment` and `Observation.` The association end `Assessment.observationCollection(Many to Many)` is visible. The table `ASSESSMENT_OBSERVATION` is the correlation-table. Then, drag the association end `(Assessment.ObservationCollection (Many to Many))` and drop it onto the corresponding column (`OBSERVATION_ID`)) in the correlation table (`ASSESSMENT_OBSERVATION`).

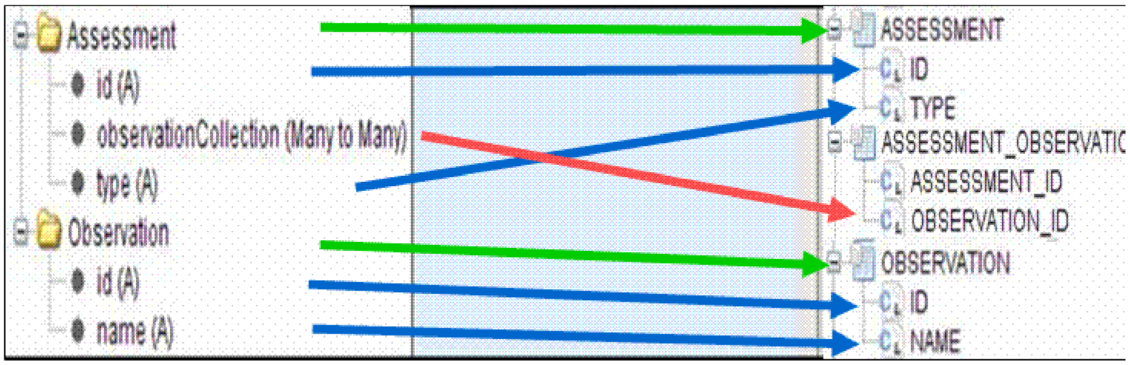


Figure 2.19 Many-to-Many Uni-Directional Mapping

Polymorphism and Inheritance Mapping

Hibernate object-relational mapping and persistence framework provides a lot of advanced features, ranging from introspection to polymorphism and inheritance mapping. It has proved difficulty to map class hierarchies to a relational database model. caAdapter Model Mapping Service support the following three basic Hibernate mapping strategies:

- Table per class hierarchy (see *Table Per Class Hierarchy* on page 26)
- Table per subclass (see *Table Per Subclass* on page 27)
- Table per concrete class (see *Table Per Concrete Class* on page 29)

It is possible to use different mapping strategies for different branches of the same inheritance hierarchy, and then make use of implicit polymorphism to achieve polymorphism across the whole hierarchy.

Table Per Class Hierarchy

This strategy uses a single table to store the entire class hierarchy. The super class and all subclasses are mapped to the same table. The table contains an additional column, DISCRIMINATOR. The value of this column is assigned to each subclass as is "discriminator-value". Hibernate uses this column to automatically instantiate the appropriate subclass and populate it accordingly.

The following mapping rules apply to the table per class hierarchy:

- All class map to the same persistence table
- The persistence table contains DISCRIMINATOR column
- Each subclass is assigned with unique "discriminator-value"
- Map all attributes of the super class to the persistent table
- Do not map the inherited attribute of any subclass.

In the following example, all classes in the hierarchy is persisted with the same table `SHOES`. The class `Shoe` is the super class with two attribute: id and color. The attribute `Shoe.id` is column `SHOES.ID` and attribute `Shoe.color` is mapped to column `SHOES.COLOR`. The super class `Shoe` is inherited by two subclasses:

DesignerShoes and SportShoes. The subclass has two inherited id and color attributes, but they are not mapped. The subclass DesignerShoes has association designer to object Designer, which is mapped to the foreign key column SHOES.DESIGNER_ID. The subclass SportShoes has additional attribute sportType which is mapped to column SHOES.SPORT_TYPE.

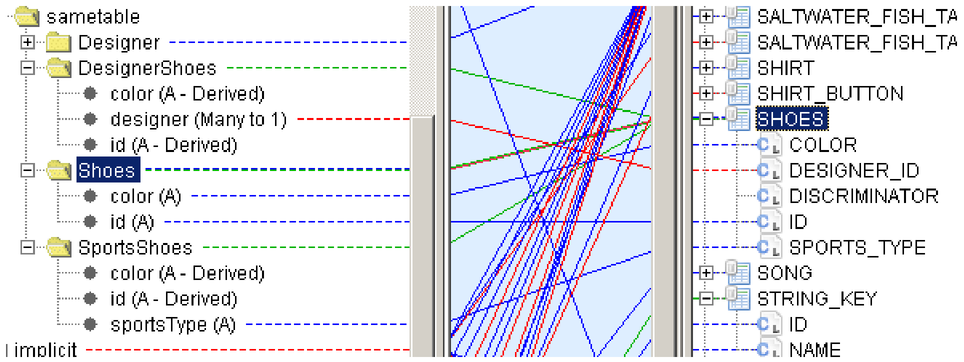


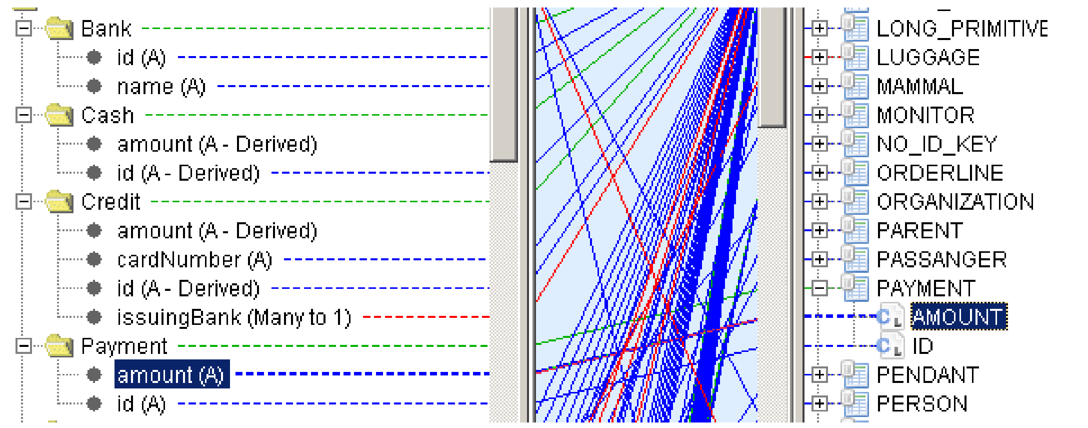
Table Per Subclass

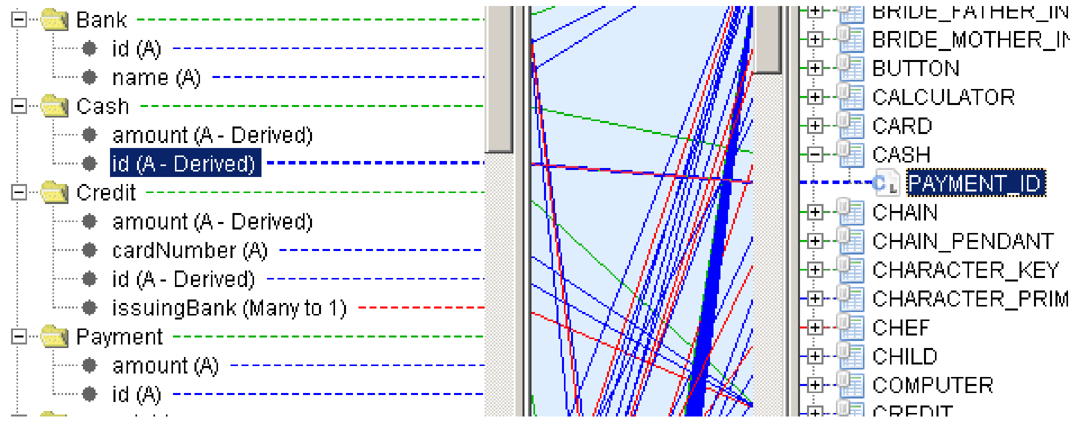
This strategy uses a different table for each class (super class and all subclasses) in the hierarchy. Each class is mapped its attributes to its own persistence table and all its associations to the class/object associated. In this strategy, the data model is very close to object model. But the data integrity requires that all these table share the same primary key. Hibernate uses this primary key when it inserts new records int database. It also uses the same primary key to perform “JOIN” operation when it accesses database.

The following mapping rules apply to the Table Per Subclass:

- Each class maps to the individual persistence table.
- All these table share the same primary key.
- Each class maps its own attributes and associations.
- All subclass must map its primary key attribute even it is an inherited attribute.
- Do not map the other inherited attributes of any subclass.

In the following example, the super class and subclasses in the hierarchy are persisted to separated tables. The class Payment is the super class with two attributes: id and amount. The attribute Payment.id is column PAYMENT.ID and attribute. Payment.amount is mapped to column PAYMENT.AMOUNT. The subclasses do not map the inherited attribute `amount` but map attribute `id` since it is primary key. The subclass `Credit` maps its additional attribute `Credit.cardNumber` to column `CREDIT.CARD_NUMBER` and its association `Credit.issuingBank` to the foreign key column `CREDIT.BANK_ID`.





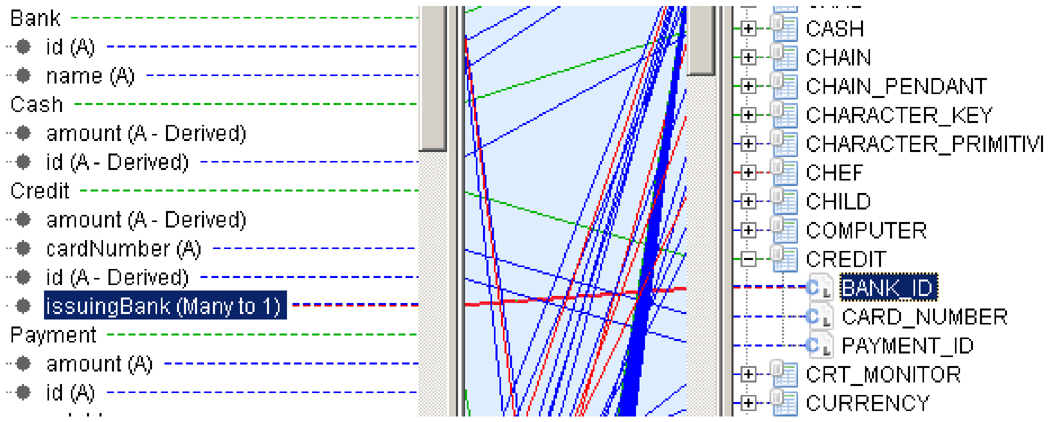


Table Per Concrete Class

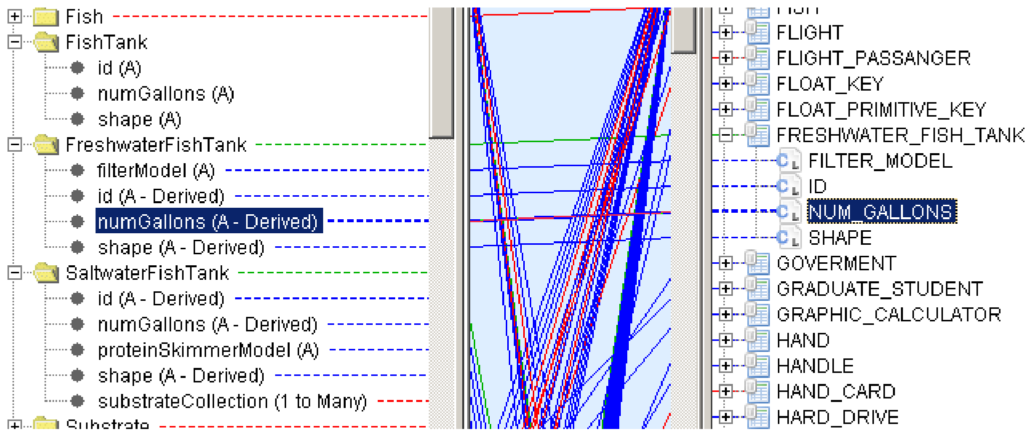
This strategy uses one table per concrete class and none for the abstract class. All tables share the same primary key, which will never allow the identical primary key values shared between two tables. For each concrete class, it maps all its attributes (including inherited properties) to its persistence table. The super class does not appear in the mapping, but exists since all subclasses extend from it.

In this strategy, Hibernate uses `Implicit Polymorphism` and “introspection” to identify the classes that extend from super abstract class and perform the appropriate SQL for each of the subclasses.

The following mapping rules apply to the Table per Concrete Class hierarchy:

- Do not map super class to any persistence table.
- Each subclass is mapped to its own persistence table.
- All these tables share the same primary key.
- All attributes (including inherited attribute) of a subclass are mapping to their persistence table.

In the following example, the super class is `FishTank`. The subclasses are `FreshwaterFishTank` and `SaltwaterFishTank`. The super class is not mapped. The subclasses map their attributes to their persistence tables: `FRESHWATER_FISH_TANK` and `SALTWATER_FISH_TANK`.



Note: `Implicit Polymorphism` means that instances of the class will be returned by query that names any super class or implemented interface or that class. Instances of that class and any subclass will be returned by a query that names that class itself.

`Introspection` is a capability of some object-oriented programming languages to determine the type of an object at runtime. It is the foundation for implementing `polymorphism`.

Additional Operations on Object Model

Object Discriminator Value

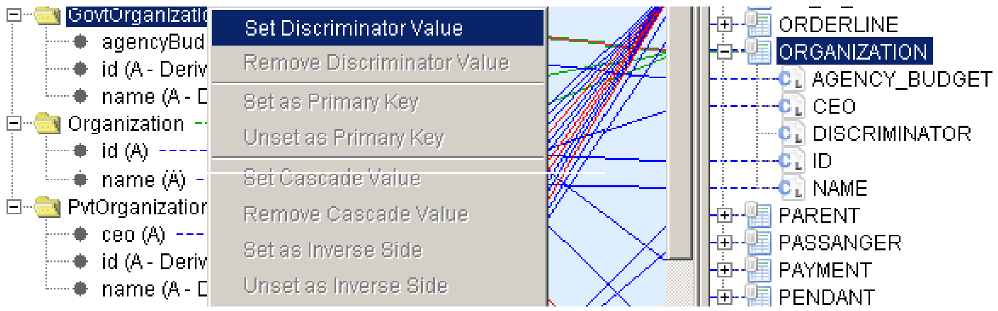
In the Table per Hierarchy Hibernate mapping strategy, the super class and all its subclasses are mapped to the same persistence table. The persistence table must contain an additional column, `DISCRIMINATOR`. Each subclass is assigned a unique discriminator value. Hibernate uses this discriminator value to determine how to instantiate the appropriate class for each row of the persistence table. caAdapter Model Mapping Service allows users to set, update, and delete the discriminator value assigned to a class.

Set a New Discriminator Value

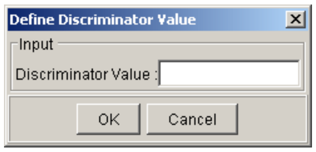
1. Right-click an attribute on an object in the left panel. A popup window appears.

   The Set Discriminator Value menu option is enabled in this popup window if the following is true:

   - the same class hierarchy is mapped to the same persistence table
   - the selected class is a subclass
   - no discriminator has been assigned to the selected class



2. Select the **Set Discriminator Value** menu option. The Define Discriminator Value window appears, in which you can specify your discriminator for the selected class.



3. Enter a value in the Discriminator Value box.

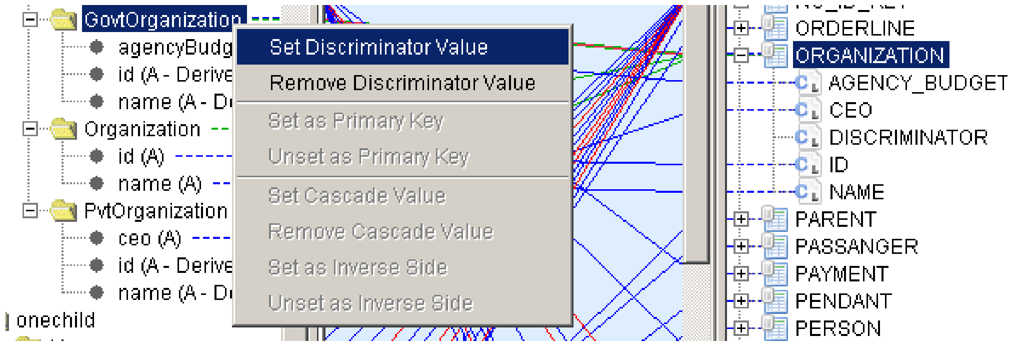
   Note: The discriminator value must be unique in the class hierarchy.

4. Click **OK**. The value you entered is now the discriminator value for the selected class.

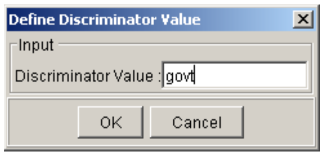
Change a Discriminator Value

1. Right-click an attribute on an object in the left panel. A popup window appears.

   If you have previously set a discriminator value on this object, the Set Discriminator Value and Remove Discriminator Value menu options are both enabled.



2. Select **Set Discriminator Value**. The Define Discriminator Value dialog box appears, showing the current discriminator value assigned to the selected class.

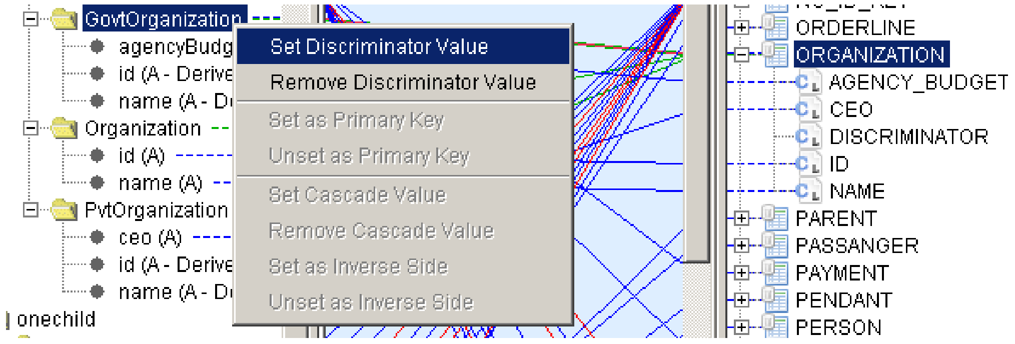


Discriminator Value : govt

3. In the Discriminator Value box, enter a new value.
4. Click **OK**. The new value is assigned to the selected class as its discriminator value.

Remove an Existing Discriminator Value

1. Right-click an attribute on an object in the left panel. A popup window appears.



2. Select **Remove Discriminator Value**. The discriminator value is no longer assigned to the selected class.

Primary Key Attribute

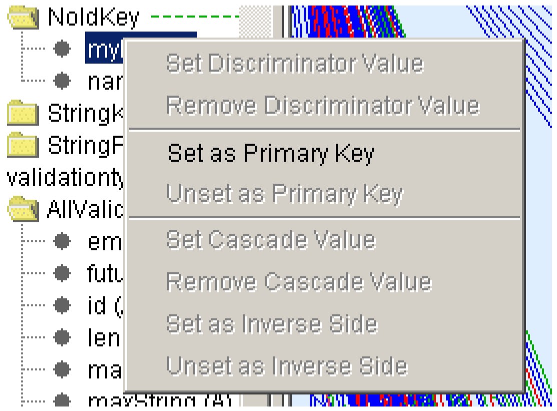
caCORE SDK requires an attribute is marked as the class `identifier` attribute when the identifier attribute is named something other than the default name, *`id`*. caAdapter Model Mapping Service allows you to set and unset an attribute as the class `identifier` attribute.

Set as Primary Key

1. Right-click an attribute on a class in the left panel. A popup window appears.
2. Select **Set as Primary Key**. The attribute you selected is set as the class `identifier` attribute for the selected class.

   Note that the Set as Primary Key menu option is only enabled if

   - the selected class attribute is mapped to a table column.
   - no attribute is set as the class `identifier` attribute for the selected class

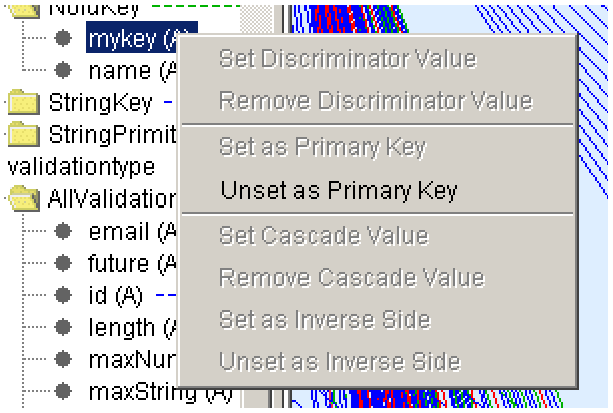


Note: Only one attribute can be set as class `identifier` attribute for any class.

Unset as Primary Key

1. Right-click an attribute on a class in the left panel. A popup window appears.
2. Select **Unset as Primary Key**.

   The Unset as Primary Key menu is only enabled if the attribute you selected was previously set as the primary key.



The selected attribute is removed as the class `identifier` attribute, or primary key, for the selected class.

Association Cascade Setting

In an Hibernate object-relational mapping, the cascade flag specifies which operations should cascade from the parent object to the associated object. Hibernate supports the following list of cascade-type values:

- none
- save-update
- delete
- all
- all-delete-orphan
- delete-orphan

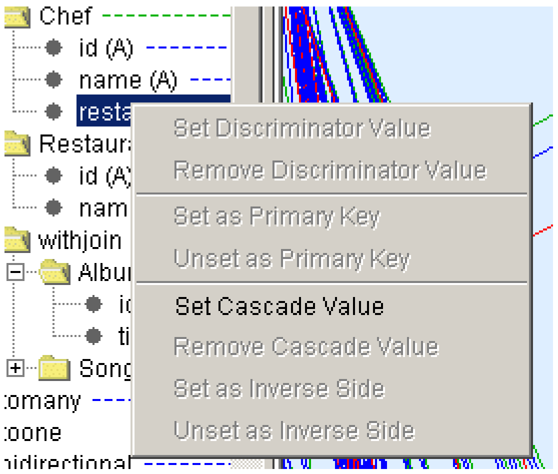
During code generation process, the caCORE SDK reads the UML tag values specified on an association end, indicating which cascading style to use. If you do not specify any cascade style, the SDK generates a Hibernate file without a cascade style. In this case, the association will not be updated when an object is persisted. The caAdapter Model Mapping Service allows you to set, update, and delete a cascade setting value for any association end.

Assign a New Cascade Value

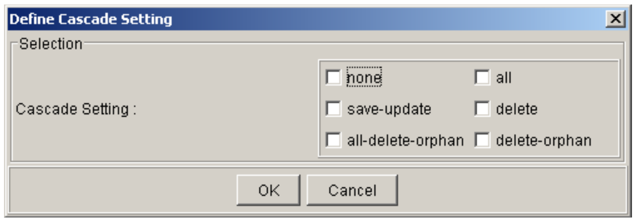
1. Right-click an association end on a class in the left panel. A popup window appears.

   The Set as Cascade Value menu is enabled if

   - the selected association end is mapped to a table column
   - no cascade setting value is assigned to the selected association end



2. Select **Set Cascade Value**. A window appears that shows all permissible values as check boxes.



Cascade Setting : ☐ none ☐ all ☐ save-update ☐ delete ☐ all-delete-orphan ☐ delete-orphan

3. Select the check box with the preferred value.

4. Click **OK**. The selected value is now the class cascade style for the selected association end.

Update an Existing Cascade Setting

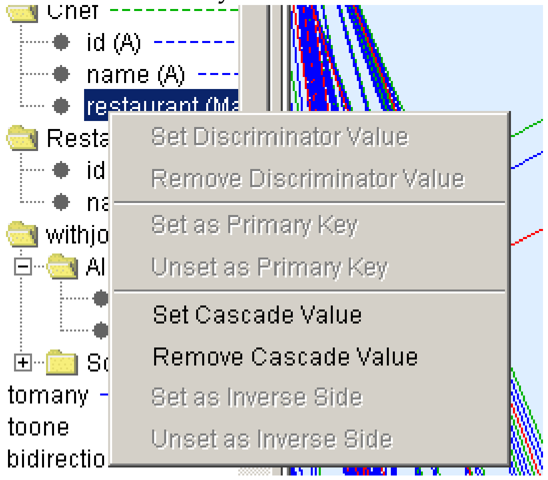
1. Right-click an association end on a class in the left panel. A popup window appears.

   The Set as Cascade Value menu option is enabled if

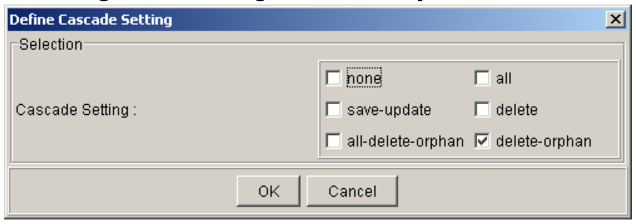
   - the selected association end is mapped to a table column, and
   - no cascade setting value is associated with the selected association end

2. Select **Set Cascade Value**. A window appears that shows all permissible values as check boxes.

   Note: Both the Set as Primary Key and Remove Cascade Value menu options are enabled if the selected association end was previously set with a cascade-style value.



3. Click **Set Cascade Value**. The Define Cascade Setting dialog box appears showing the existing cascade style value.



Cascade Setting : ☐ none ☐ all ☐ save-update ☐ delete ☐ all-delete-orphan ☑ delete-orphan

4. Select a new cascade setting.

5. Click **OK**.

Unset Cascade Value

1. Right-click an association end on a class in the left panel. A popup window appears.

   The Set as Cascade Value menu option is enabled if

   - the selected association end is mapped to a table column, and
   - no cascade setting value is associated with the selected association end

2. Select **Set Cascade Value**. A window appears that shows all permissible values as check boxes.

   Note: Both the Set as Primary Key and Remove Cascade Value menu options are enabled if the selected association end has previously been set with a cascade style value.

3. Click the **Remove Cascade Value** menu. The cascade style value assigned to association end is removed.

Inverse Association Side

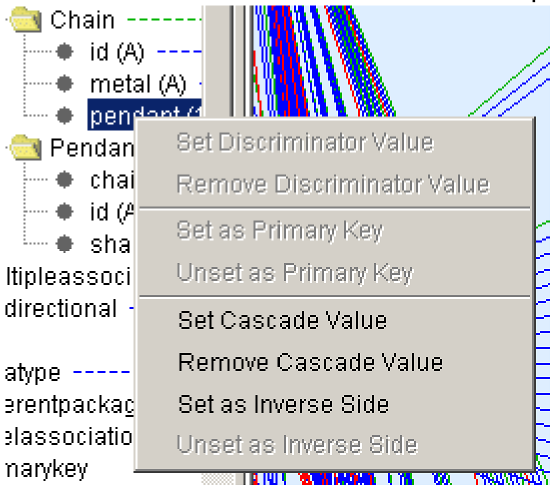
In an Hibernate object-relational mapping, the inverse flag indicates which end of an association should be ignored as persist this association. For a bi-directional association, the caCORE SDK requires you to mark one end of the association as the inverse side. The caAdapter Model Mapping Service allows you to set and unset an association end as the inverse side.

Set as Inverse Side

1. Right-click an attribute on a class in the left panel. A popup menu appears.

   The Set as Inverse Side menu option is enabled if

   - the selected association end is mapped to a table column
   - correlation table is used in the association mapping
   - the selected association end was not previously set as inverse side



2. Select **Set as Inverse Side**. The selected end is set as the association inverse side.

Note: Only one end of an association can be set as the inverse side. If you change the inverse side, the other side is automatically cleared of the inverse side setting.

Unset as Inverse Side

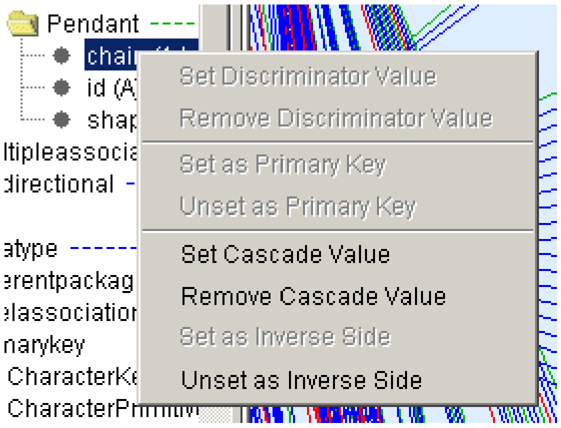
1. Right-click an attribute on a class in the left panel. A popup menu appears.

   The Set as Inverse Side menu is enabled if

   - the selected association end is mapped to a table column
   - correlation table is used in the association mapping
   - the selected association end was not previously set as inverse side

2. Select **Set as Inverse Side**. The selected end is set as the association inverse side.

   The Unset as Inverse Side menu option is enabled if the selected association end was previously set as the association inverse side.



4. Select **Unset as Inverse Side**. The selected association end is removed as the inverse side.

Additional Operations on Data Model

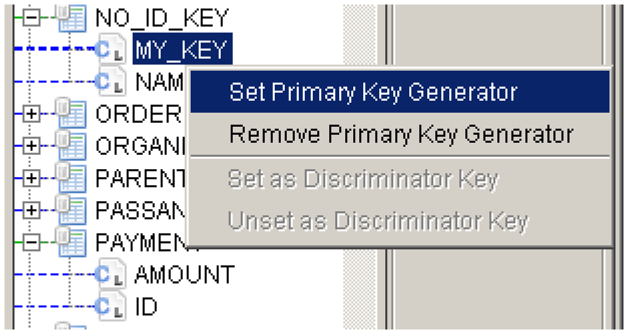
Primary Key Generator

The caCORE SDK allows you to specify the primary key at two levels: global or individual class level. The *caCORE SDK User's Guide* provides detailed procedures on how to specify the global primary key generator. The caAdapter Model Mapping Service allows you to set, update, and delete the primary key generator with a particular class.

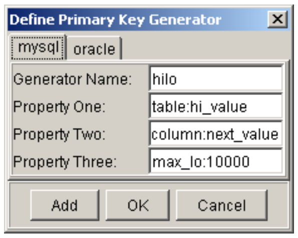
Define a new Primary Key Generator

1. Right click a column on the table in the right panel. A popup menu appears.

   The Set Primary Key Generator menu is enabled if this column is a primary key column.

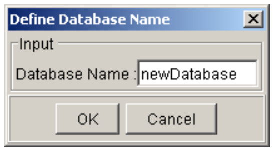

NO_ID_KEY
MY_KEY
NAM
ORDER
ORGANI
PARENT
PASSAN
PAYMEN
AMOUNT
ID
Set Primary Key Generator
Remove Primary Key Generator
Set as Discriminator Key
Unset as Discriminator Key

2. Select **Set Primary Key Generator**. The Define Primary Key Generator window appears.



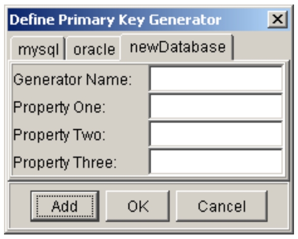
Generator Name: hilo
Property One: table:hi_value
Property Two: column:next_value
Property Three: max_lo:10000

3. Click the **Add** button. The Define Database Name window appears.



Database Name : newDatabase

4. Type the name of the database and click **OK**. A new tab is added for the new database with fields to collect required properties.



5. Enter values for each property and then click **OK**. A new primary key generator is added to the selected table.

Update a Primary Key Generator

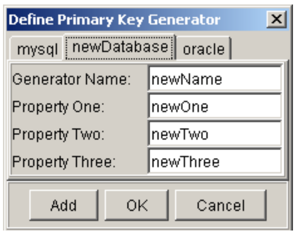
1. Right click a column on the table in the right panel. A popup menu appears.

   The Set Primary Key Generator menu is enabled if this column is a primary key column.

2. Select **Set Primary Key Generator**. The Define Primary Key Generator window appears.

3. Click the **Add** button. The Define Database Name window appears.

4. Type the name of the database and click **OK**. A new tab is added for the new database with fields to collect required properties
5. Enter values for each property and then click **OK**. A new primary key generator is added to the selected table.
6. Select the primary key generator tab.



Generator Name: newName
Property One: newOne
Property Two: newTwo
Property Three: newThree

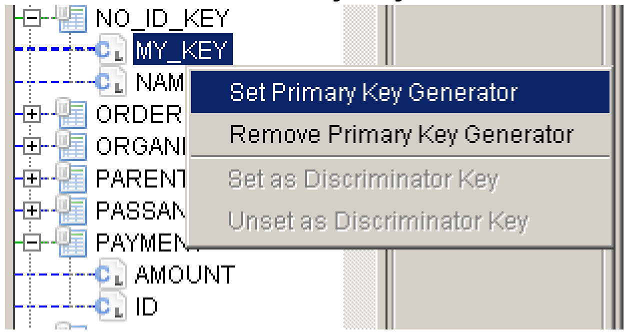
7. Change the value of any property.
8. Click **OK**. You can now update the properties of the primary key generator for a selected table.

Remove an Existing Primary Key Generator

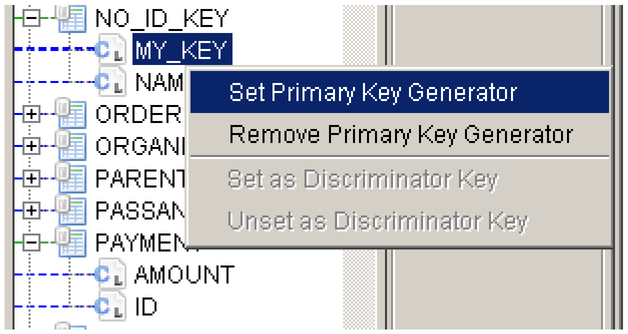
1. Right click a column on the table in the right panel. A popup menu appears.

   The Remove Primary Key Generator menu is enabled if a primary key generator has already been specified for the selected table.

2. Click the **Remove Primary Key Generator** menu.


NO_ID_KEY
MY_KEY
NAM
ORDER
ORGANI
PARENT
PASSAN
PAYMEN
AMOUNT
ID
Set Primary Key Generator
Remove Primary Key Generator
Set as Discriminator Key
Unset as Discriminator Key

A popup menu appears that lists all existing primary key generators with the selected table.


NO_ID_KEY
MY_KEY
NAM
ORDER
ORGANI
PARENT
PASSAN
PAYMEN
AMOUNT
ID
Set Primary Key Generator
Remove Primary Key Generator
Set as Discriminator Key
Unset as Discriminator Key

3. Select the name of a primary key generator from the list.
4. Click **OK**. The selected primary key generator is removed from the selected table.

Discriminator Key Column

In the Table per Hierarchy Hibernate mapping strategy, the super class and all its subclasses are mapped to the same persistence table. The persistence table is required to contain an additional column, `DISCRIMINATOR`. Hibernate uses this column to automatically instantiate the appropriate class and populate accordingly. The caAdapter Model Mapping Service allows you to set and unset the `DISCRIMINATOR` column if required.

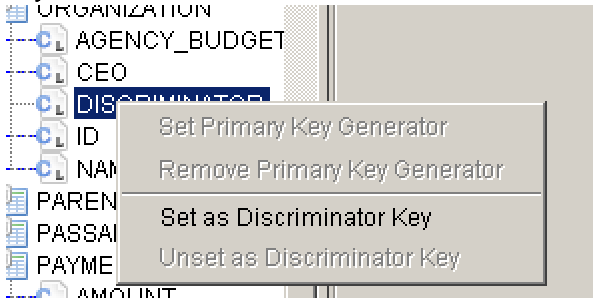
Set as Discriminator Key

1. Right-click a column on a table in the right panel. A popup menu appears.

   The Set as Discriminator Key menu option is enabled if

   - the table is mapped to support a whole multiple classes in the same hierarchy
   - the select column is not mapped to any `object.attribute`

2. Select **Set as Discriminator Key**. The selected column is now the discriminator key.



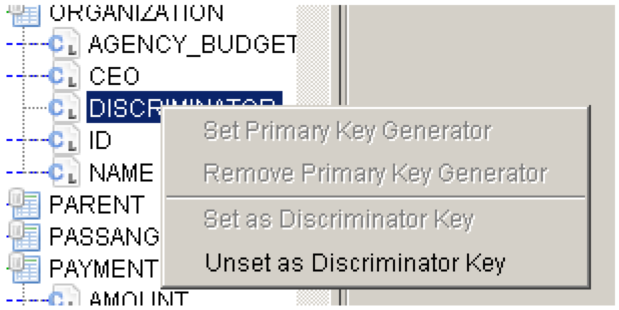
Note: Only one column per table can serve as the discriminator.

Unset as Discriminator Key

1. Right-click a column on a table in the right panel. A popup menu appears.

   The Unset as Discriminator Key menu is enabled if the selected column was being previously set as the discriminator column.

2. Select **Unset as Discriminator Key**. The column is no longer the discriminator and is available to map to any `object.attribute`.



User Interface Legend

Node Details

- `(A)` – The node is an attribute
- `(A – Derived)` –The node is an inherited attribute
- `(1 to 1)` – The node is a one-to-one association
- `(1 to Many)` – The node is a-one-to many association
- `(Many to 1)` – The node is a many-to-one association
- `(Many to Many)` – The node is a many-to-many association

Mapping Line Colors

- Green – Dependency Mapping
- Blue – Attribute Mapping
- Red – Association Mapping

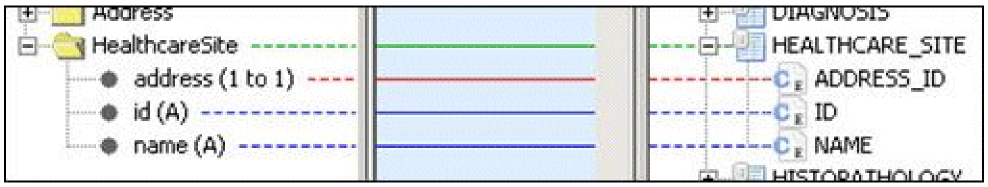
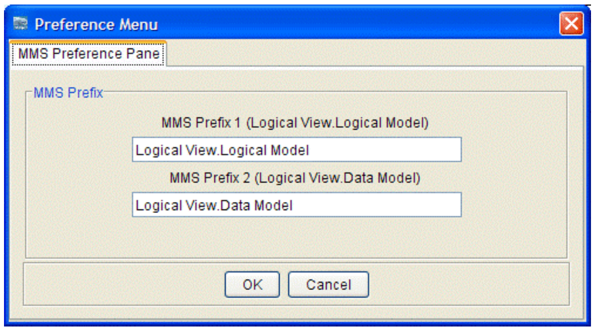


Figure 2.20 Mapping Line Colors

Prefix for Object and Data Model

Specify the prefix to use for object or data model elements using the Tools > Preferences menu option (*Figure 2.21*).



MMS Prefix 1 (Logical View.Logical Model)
Logical View.Logical Model
MMS Prefix 2 (Logical View.Data Model)
Logical View.Data Model

Figure 2.21 Designating Prefixes on the Preference Menu

The following table shows the tags that caAdapter implements.

Tag Name	***Tag Value***	***Location***
id-attribute	Fully qualified class name	Class attribute

Figure 2.22 caAdapter Implemented Tags

Tag Name	***Tag Value***	***Location***
mapped-attributes	Fully qualified attribute name	Table column
implements-association	Fully qualified association name	Table column (foreign key)
discriminator	Fully qualified class name (when used on the column), Discriminating value (when used on the class)	Table column (foreign key), Class

Figure 2.22 caAdapter Implemented Tags

Example Data Files

Example data are included in the caAdapter MMS Tool distribution. You can use the example data to become acquainted with the mapping tool before using your own data. Example data are located at the {home directory}\workingspace\.

APPENDIX A REFERENCES

Articles

- Java Programming: http://java.sun.com/learning/new2java/index.html
- Extensible Markup Language: http://www.w3.org/TR/REC-xml/
- XML Metadata Interchange: http://www.omg.org/technology/documents/formal/xmi.htm

caBIG Material

- caBIG: http://cabig.nci.nih.gov/
- caBIG Compatibility Guidelines: http://cabig.nci.nih.gov/guidelines_documentation

caCORE Material

- NCI CBIIT: http://ncicb.nci.nih.gov
- caCORE: https://cabig.nci.nih.gov/tools/caCORE_SDK
- caBIO: https://wiki.nci.nih.gov/display/ICR/caBIO
- caDSR: http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/cadsr

Software Products

- Java: http://java.sun.com
- Ant: http://ant.apache.org/

caAdapter Glossary

Acronyms, objects, tools and other terms related to the caAdapter MMS Tool are described in this glossary.

Term	***Definition***
caCORE SDK	cancer Common Ontologic Representation Environment Software Development Kit
CLOB	Character large object
EA	Enterprise Architect
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	Extensible Markup Language

INDEX