# caAdapter Model Mapping Service (MMS) 4.4

## *User's Guide*


NATIONAL® CANCER INSTITUTE

Center for Biomedical Informatics and Information



June 4, 2010 3:55 am

# CONTENTS


**About This Guide ..... 1**

Purpose ..... 1
Audience ..... 1
Typical User ..... 1
Prerequisites ..... 1
Organization of This Guide ..... 2
Recommended Reading ..... 2
Text Conventions Used ..... 3
Credits and Resources ..... 3

**Chapter 1**
**Overview of caAdapter ..... 5**

About caAdapter ..... 5
caAdapter Core Engine Architecture ..... 6
caAdapter Mapping Tool ..... 7
About HL7 ..... 7
About the Study Data Tabulation Model (SDTM) ..... 9
About the Object and Data Model ..... 10
Prerequisites for Using the caAdapter Mapping Tool ..... 11
Resources for Installing caAdapter MMS Tool ..... 11
Starting the caAdapter MMS Tool ..... 11
Starting the caAdapter MMS Tool from the Binary Distribution ..... 11
Starting the caAdapter MMS Tool from the Source Distribution ..... 11
Starting the caAdapter MMS Tool from the Windows Distribution ..... 11
Starting the Mapping Tool on the Web (WebStart) ..... 11

**Chapter 2**
**Using the caAdapter MMS Tool ..... 13**

About the caAdapter Model Mapping Service ..... 13
Operational Scenario for the Model Mapping Service ..... 14
Using the caAdapter Model Mapping Service ..... 14
Generating an XMI File from EA ..... 15

Creating an Object Model to Data Model Map Specification ..... 16
Opening an Existing Object to Data Model Mapping Specification ..... 17
Validating Mapping Specifications ..... 17
Saving Mapping Specifications ..... 18
Generating Hibernate Object-Relational Mapping Files ..... 19
Understanding Basic Mapping ..... 19
Dependency Mapping (Object to Table) ..... 20
Attribute Mapping ..... 21
Association Mapping ..... 21
Deleting Mapping Lines ..... 22
Understanding ISO 21090 Data Type Mapping ..... 22
Map Simple Data Type Attribute ..... 22
Map Complex Data Type Attribute ..... 22
*Map Complex Data Type Attribute to Parent Class Table ..... 23*
*Map Complex Data Type Attribute to a Separate Table ..... 23*
Set Global Constant ..... 24
Remove Global Constant ..... 25
Set Local Constant ..... 25
Remove Local Constant ..... 26
Set Global NullFlavor Constant ..... 26
Remove Global NullFlavor Constant ..... 27
Set Local NullFlavor Constant ..... 28
Remove Local NullFlavor Constant ..... 29
Map Complex Data Type with Collection Attribute ..... 29
Map Collection Data Types without a Joint Table ..... 30
Map Collection Data Types with a Joint Table ..... 31
Map Collection Data Type with a Collection Attribute ..... 32
Set Collection Element Type ..... 33
Remove Collection Element Type ..... 34
Map to Implicit Key ..... 34
Unmap From Implicit Key ..... 35
Set as Inverse Side ..... 36
Unset as Inverse Side ..... 36
Understanding the Seven Association Mapping Scenarios ..... 37
One-to-One Bi-Directional ..... 37
One-to-One Uni-Directional ..... 37
One-to-Many/Many-to-One Bi-Directional ..... 38
Many-to-One Uni-Directional ..... 39
One-to-Many Uni-Directional ..... 39

Many-to-Many Bi-Directional ..... 40
Many-to-Many Uni-Directional ..... 41
Understanding Polymorphism and Inheritance Mapping ..... 42
Table Per Class Hierarchy ..... 42
Table Per Subclass ..... 43
Table Per Concrete Class ..... 45
Additional Operations on Object Model ..... 45
Object Discriminator Value ..... 45
*Set a New Discriminator Value ..... 46*
*Change a Discriminator Value ..... 47*
*Remove an Existing Discriminator Value ..... 47*
Primary Key Attribute ..... 47
*Set as Primary Key ..... 48*
*Unset as Primary Key ..... 48*
Association Cascade Setting ..... 49
*Assign a New Cascade Setting Value ..... 49*
*Update an Existing Cascade Setting ..... 50*
*Unset Cascade Value ..... 51*
Inverse Association Side ..... 51
*Set as Inverse Side ..... 51*
*Unset as Inverse Side ..... 52*
Additional Operations on Data Model ..... 52
Primary Key Generator ..... 52
*Define a New Primary Key Generator ..... 53*
*Update a Primary Key Generator ..... 54*
*Remove an Existing Primary Key Generator ..... 54*
Discriminator Key Column ..... 55
*Set as Discriminator Key ..... 55*
*Unset as Discriminator Key ..... 55*
User Interface Legend ..... 56
Node Details ..... 56
Mapping Line Colors ..... 56
Prefix for Object and Data Model ..... 57
Example Data Files ..... 57

**Appendix A**
**References ..... 59**

Articles ..... 59
caBIG Material ..... 59
caCORE Material ..... 59
Software Products ..... 59

**caAdapter Glossary .....61**

**Index .....63**

# About This Guide

This section introduces you to the *caAdapter MMS 4.4 User's Guide*.

Topics in this section:


- *Purpose* on this page
- *Audience* on this page
- *Organization of This Guide* on page 2
- *Recommended Reading* on page 2
- *Text Conventions Used* on page 3
- *Credits and Resources* on page 3


## 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 open source caAdapter tool set, which provides model mapping services and facilitates data mapping and transformation services.

## Audience

### Typical User

This guide is designed for users who want to map a data model to an object model and build 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 4.4 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 59 provides a list of references used to produce this guide or referred to within the text.
- *caAdapter Glossary* on page 61 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 Gaheen [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] Independent Consultant |

| *Contacts and Support* | |
|---|---|
| Application Support | http://ncicb.nci.nih.gov/NCICB/support<br>Telephone: 301-451-4384<br>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 4.1.2 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* on this page
- *About HL7* on page 7
- *About the Study Data Tabulation Model (SDTM)* on page 9
- *About the Object and Data Model* on page 10
- *Prerequisites for Using the caAdapter Mapping Tool* on page 11
- *Resources for Installing caAdapter MMS Tool* on page 11
- *Starting the caAdapter MMS Tool* on page 11


## About caAdapter

caAdapter (http://ncicb.nci.nih.gov/NCICB/infrastructure/cacore_overview/caadapter) is an open source tool set that provides 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 the model mapping service (MMS), data mapping and transformation service (MTS), validation component, and other sub-components. The MMS component maps an object model to a 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). The

validation component validates messages 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 parses and loads data and object models from an XMI file, adds caCORE SDK-required tags and tag values into the XMI file, and generates Hibernate mapping files.

caAdapter integrates with NCI CBIIT cancer Common Ontologic Representation Environment (caCORE) (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.

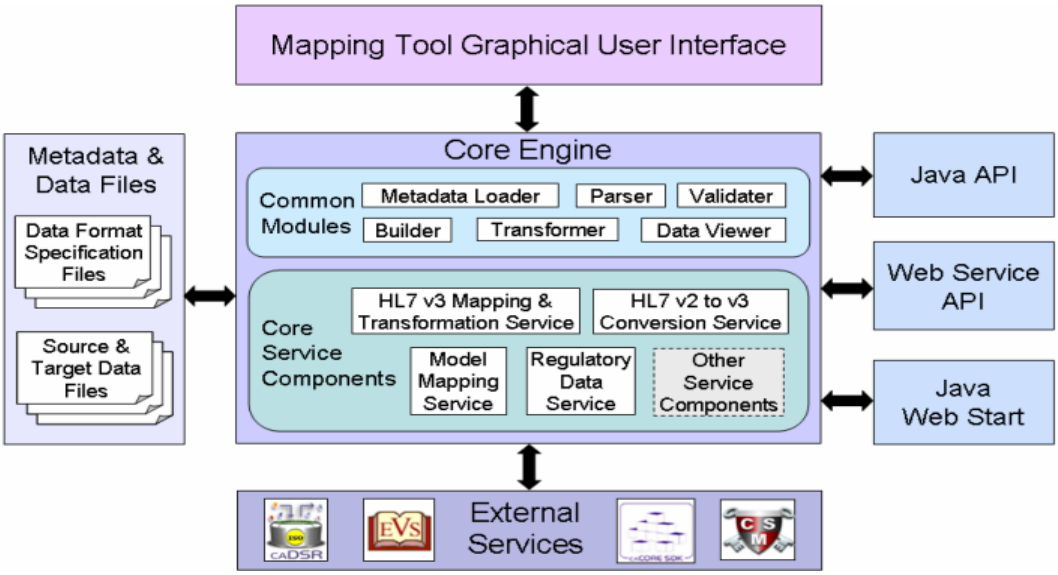



*Figure 1.1 caAdapter Core Engine Architecture*

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 user interface
- 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 - defines input and output formats.
- User Interface - maps source to target elements containing tree structure, drag-and-drop functionality, and functions and property definitions.
- Mapping Functions - manipulates source data.
- Transformation Service - generates HL7 v3 XML message instances and SDTM text files from a source database based on user-defined mapping specifications.
- Validation - validates 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
- Special Purpose

*Figure 1.2* lists SDTM Domain Classes and associated Domains.

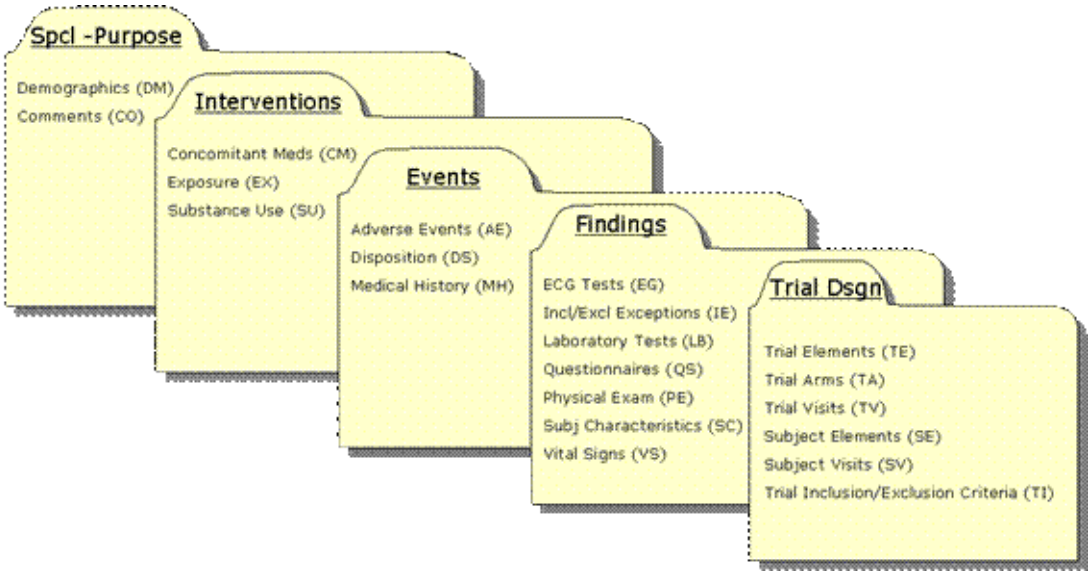



*Figure 1.2 SDTM Domain Classes*

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

An object model is the object-oriented interface to some service or system. It is a collection of objects or classes associated with properties and operations through which a program can examine and manipulate some specific parts to perform the required services. Object models are usually defined using concepts such as class, message, inheritance, polymorphism, and encapsulation.

A data model is an abstract model that describes how data is represented and accessed. Data model explicitly determines the meaning of data, which in this case is known as structured data. Data models formally define data elements and relationships among data elements for a domain of interest.

Unified Modeling Language (UML) is a standardized general-purpose modeling language. It combines the best modeling techniques from object modeling, data modeling, business modeling, and component modeling. caAdapter Model Mapping Service processes a UML model containing an object model and a data model. It sets up the object-relational mapping from an object model to a data model in building of caCORE-like application.

This guide provides detailed instruction on using the caAdapter Model Mapping Service tool.

# 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, 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 `run.bat`.

   The Welcome to the caAdapter screen appears, followed by the caAdapter MMS application.

## 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 appears, followed by the caAdapter MMS application.

## 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 the caAdapter Model Mapping Service to facilitate object-to-data model mapping.

Topics in this chapter include:


- *About the caAdapter Model Mapping Service* on this page
- *Operational Scenario for the Model Mapping Service* on this page
- *Using the caAdapter Model Mapping Service* on page 14
- *Understanding Basic Mapping* on page 19
- *Understanding ISO 21090 Data Type Mapping* on page 22
- *Understanding the Seven Association Mapping Scenarios* on page 37
- *Understanding Polymorphism and Inheritance Mapping* on page 42
- *Additional Operations on Object Model* on page 45
- *Additional Operations on Data Model* on page 52
- *User Interface Legend* on page 56
- *Prefix for Object and Data Model* on page 57


## 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. The service allows you to map from a tree node in left panel to a tree node in the right panel using 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. The UML modeling tools export UML models represented in the format of XML Metadata Interchange (XMI).

## Operational Scenario for the Model Mapping Service

The essential operational scenario is to generate caBIG silver-level compliant of caCORE-like application, using caCORE SDK tool sets. In this scenario, the 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 tool automates this process, allowing you to set up all required mapping with 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 those users who are not using the caCORE SDK but want to map an object model to a data model. In this scenario, the Model Mapping Service tool allows you to set up 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.

.

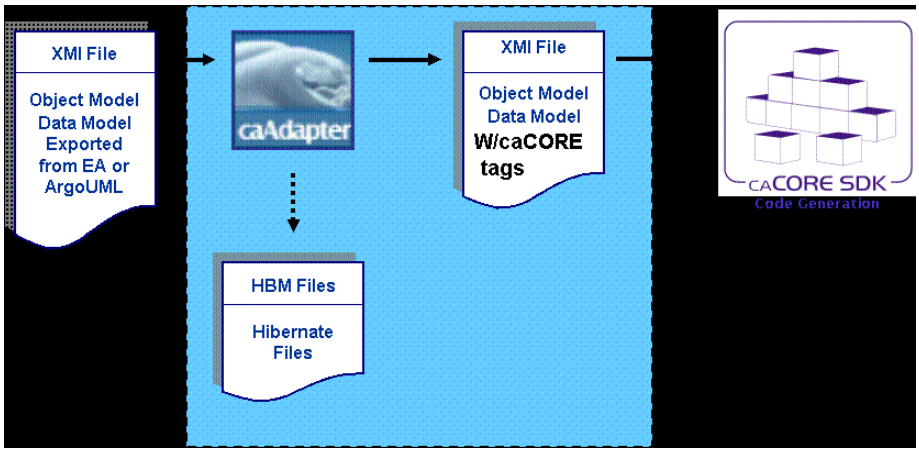



*Figure 2.1 caAdapter Model Mapping Service - Overall Process*

## Generating an XMI File from EA

Perform the following steps to export a UML model as an `XMI` file.

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.

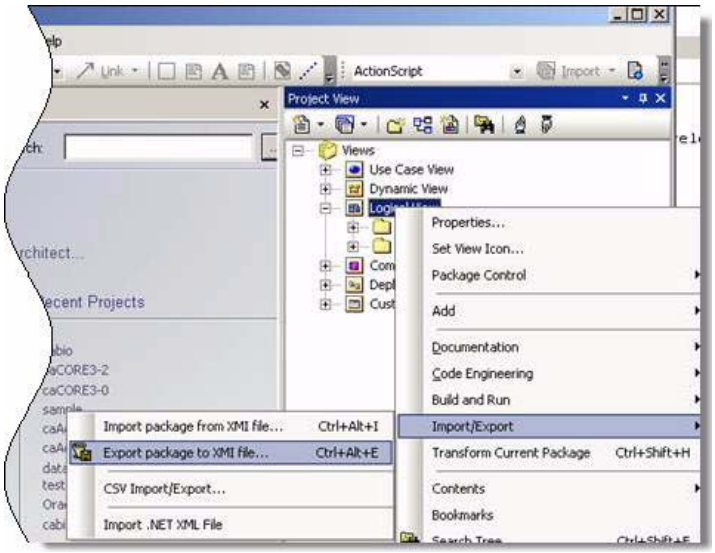



*Figure 2.2 Exporting an XMI File from EA*

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*).

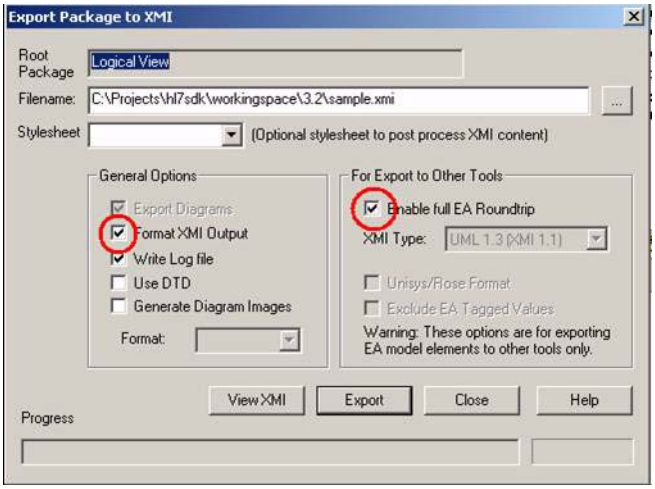



*Figure 2.3 Options to Export XMI File from EA*

## Creating an Object Model to Data Model Map Specification

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.

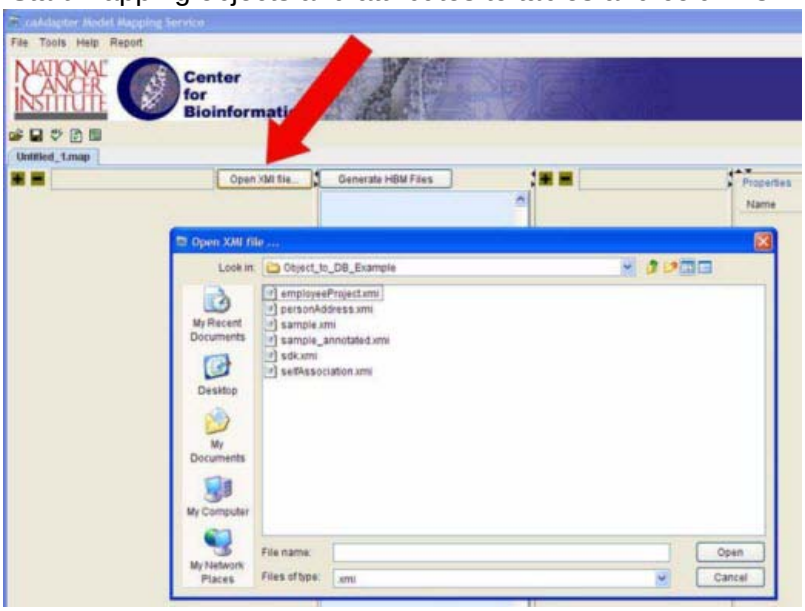



*Figure 2.4 Open XMI File button*

## 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**.

## 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 2.5*).

  If the validation process does not encounter any errors, the following message appears: `Validation process completed successfully with no message received`.

  If there are errors in the validation process, the following message appears (see *Figure 2.5*): `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.

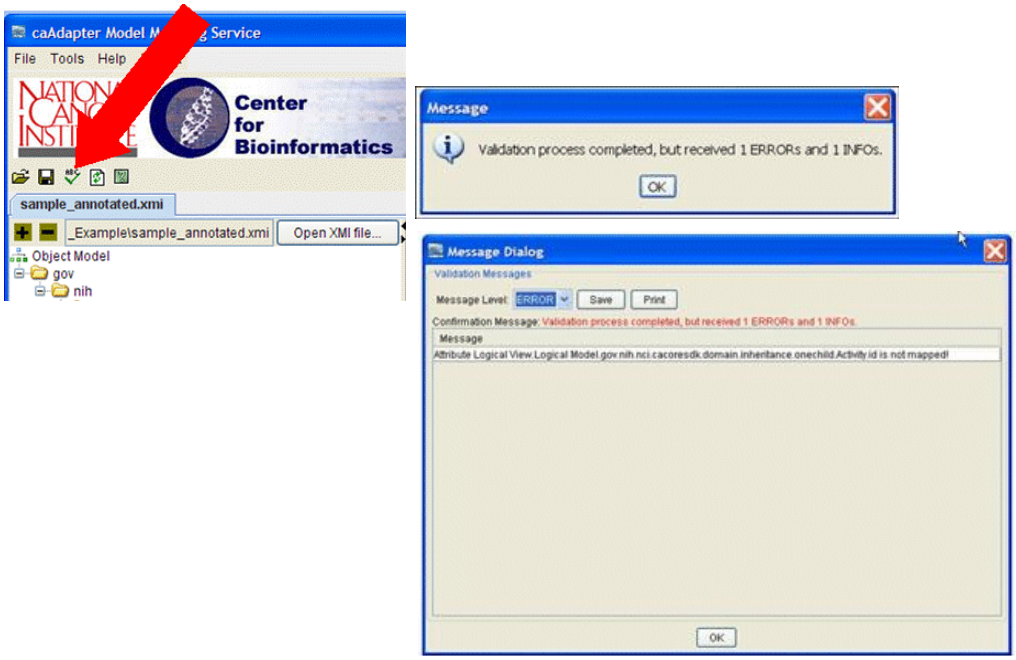



*Figure 2.5 Validate Mapping Specification*

## 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-like applications 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 2.6*).
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.

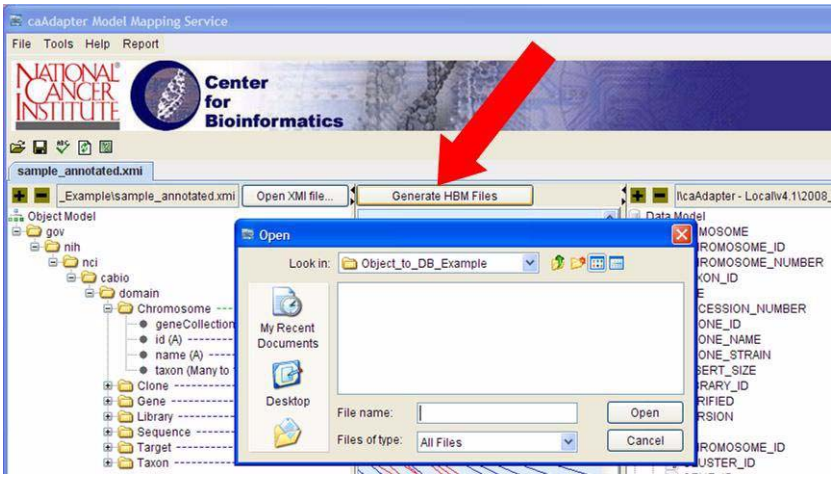



*Figure 2.6 Generate HBM Files*

# Understanding 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 2.7* shows a mapping line between `Amendment` in the object model, on the left, and `AMENDMENT` in the data model, on the right.

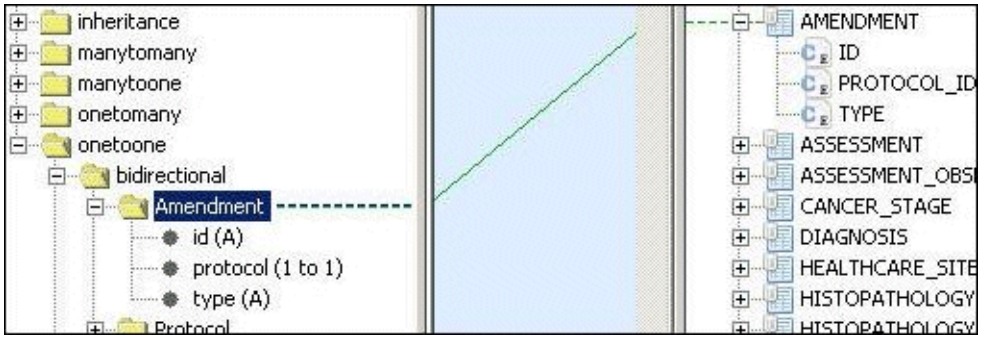



*Figure 2.7 Mapping line between source tree node and target tree node*

## 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 an object tree node from the object model on the left panel. The example in *Figure 2.8* 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.

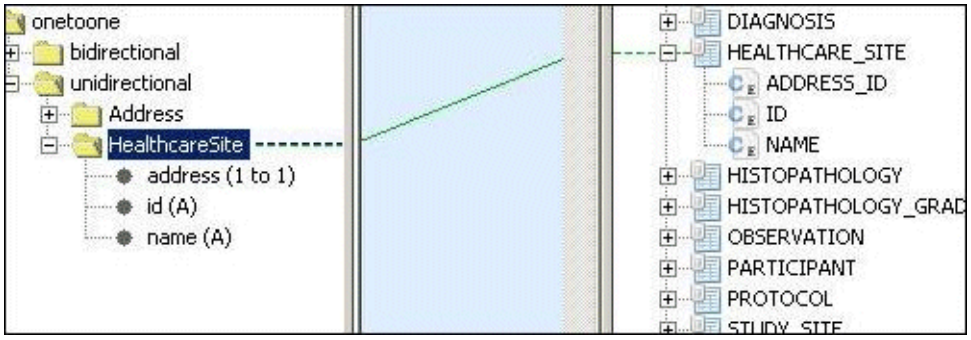



*Figure 2.8 Dependency Mapping*

**Note:** You can only map one object to one table. You can map one table from multiple objects to support the One Table Per Hierarchy inheritance mapping strategy (see *Table Per Class Hierarchy* on page 42).

## 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 2.9* 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`.

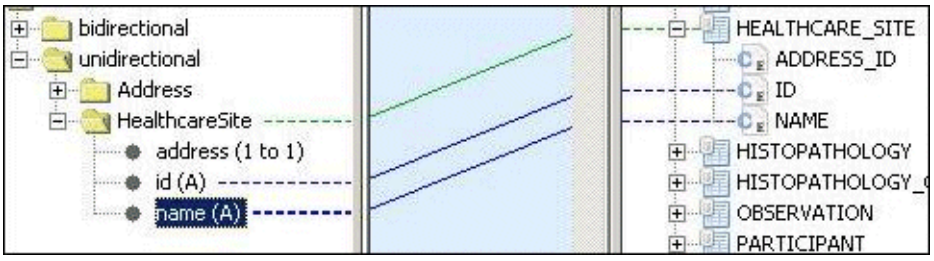



*Figure 2.9 Attribute Mapping*

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

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 2.10*).


*Figure 2.10 Attribute Mapping error message*

## 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 2.11*, 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 2.11*. Association mapping lines are color-coded red.

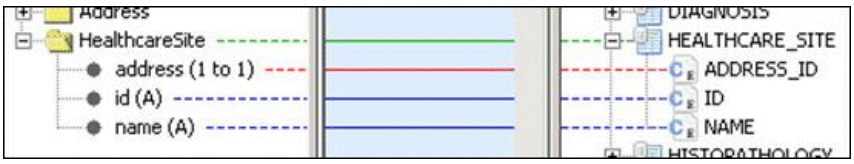



*Figure 2.11 Association Mapping*

## 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 2.12*). The line is removed from the mapping panel.

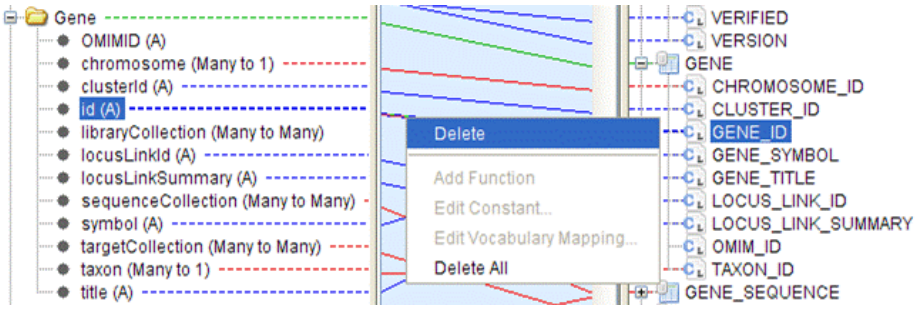



*Figure 2.12 Deleting Mapping Lines*

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

# Understanding ISO 21090 Data Type Mapping

## Map Simple Data Type Attribute

A simple data type attribute can be an attribute of a domain object or a nested attribute of complex ISO data types. A simple data type is represented by a serial of characters or a number without any structure. It includes two categories of data types. The first category is the Java primitives and wrapper types; the second category is the data types defined with ISO 21090 standard including: Code, String, Uid, Uri and XML.

For more information on mapping a simple data type to a table column, see *Attribute Mapping* on page 21.

## Map Complex Data Type Attribute

A complex data type attribute can be an attribute of a domain object or a nested attribute of complex ISO data types. Complex data types contain one or more attributes that are based on simple data types or other complex data types. Most data types defined by ISO 21090 standards are complex data types. A complex data type attribute can be mapped to one or more columns in a target table. The target table can be either the one mapped with its parent class or a separate one.

## Map Complex Data Type Attribute to Parent Class Table

This scenario maps the nested attribute of complex data types to the same target table as mapped with the parent class and all other simple data type attributes. Perform the following steps to map the nested attribute of a complex data type to the target table mapped with parent class.

1. Create the dependency mapping between the object and its target table. For example, in *Figure 2.13*, the green line shows a dependency between the object “Chef” and its target table “CHEF”.

   Object “Chef” has two attributes: “id” with complex type “II” and “name” with complex type “ST”.

2. Click and drag nested attribute “id.root” under “Chef” object in the object model panel to “ID” column of “CHEF” table. When complete, the final result should look like *Figure 2.13*.

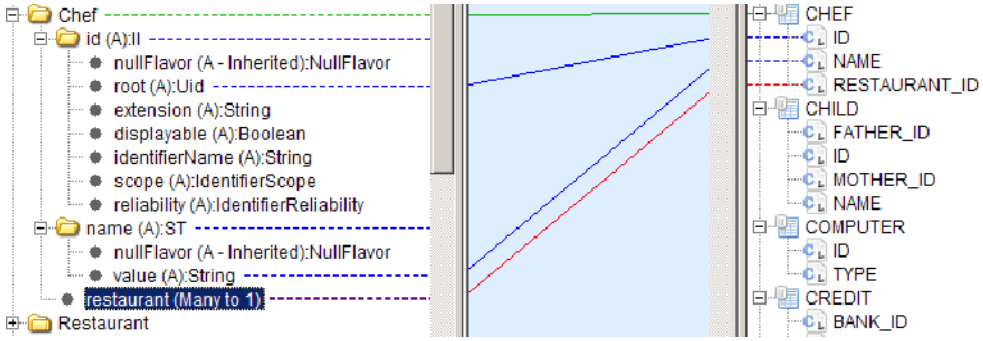



*Figure 2.13 Mapping Complex Data Type Attribute to Parent Class Table*

## Map Complex Data Type Attribute to a Separate Table

In this scenario, simple data type attributes and the nested attribute of some complex data type attributes are mapped to the same target table mapped as the parent class. But one or complex data type attributes may map their nested attributes to separate table(s). Perform the following steps to map the nested attribute of a complex data type to a separate table.

1. Create the dependency mapping between the object and its target table. For example, in *Figure 2.14*, the green line shows a dependency between the object “Pupil” and the table “PUPIL”.

   Object “Pupil” has two attributes: “id” has complex type “II” and “name” has complex type “ST”. The attribute “id” is mapped to the same table with its parent class.

2. Click and drag the nested attribute “name.nullFlavor” under “Puple” object in the object model panel to “NULL_FLAVOR” column of “STUDENT_NAME” table.

3. Repeat step 2 to map other nested attribute under “name” to other columns of the same target table “STUDENT_NAME”. When complete, the final result should look like *Figure 2.14*.

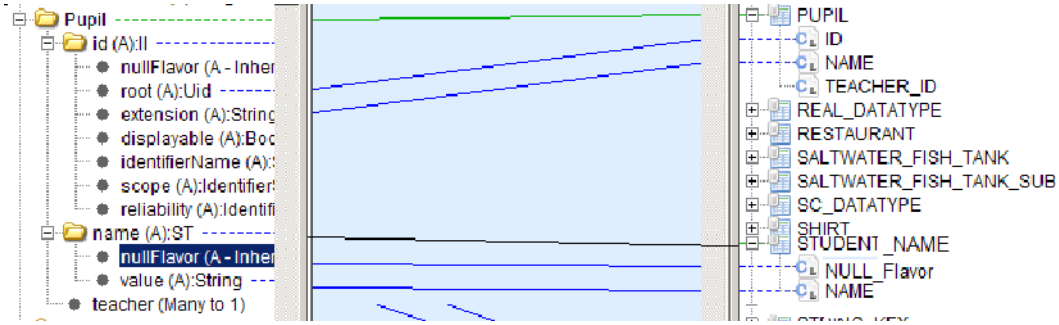



*Figure 2.14 Mapping Complex Data Type Attribute to Separate Table*

## Set Global Constant

A global constant of an attribute of ISO data types applies if the attribute is a direct attribute of an ISO object and has a simple data type. The constant value applies any instance of its parent ISO data type when the attribute is not mapped to a table column and there is no corresponding local constant value. Perform the following steps to set a global constant with a direct attribute of an ISO data type object.

1. Right-click a direct attribute node of an ISO data type object. A popup menu appears.

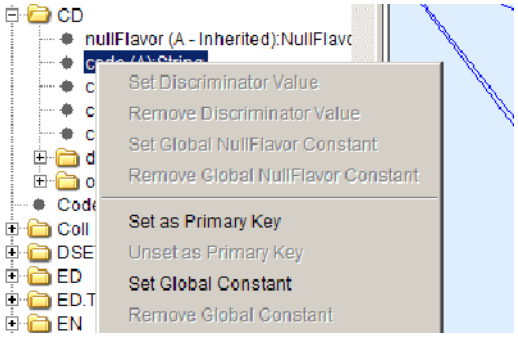



*Figure 2.15 Set Global Constant Popup*

The Set Global Constant menu is enabled if this attribute has a simple data type.

2. Select **Set Global Constant**. The Define Constant Value dialog box appears.

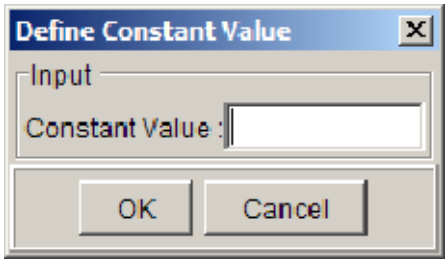



*Figure 2.16 Set Global Constant Dialog Box*

3. Enter value and then click **OK**. A new global constant is added to the selected attribute.

## Remove Global Constant

1. Right-click a direct attribute node of an ISO data type. A popup menu appears.

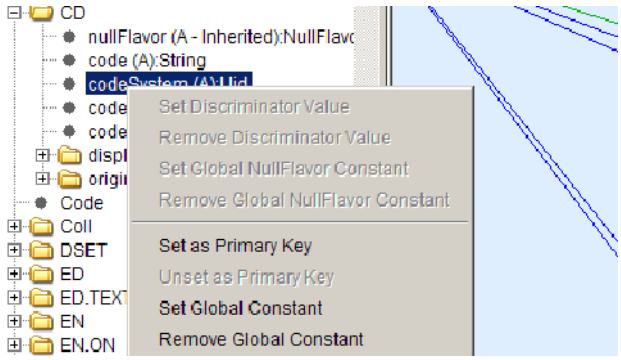



*Figure 2.17 Remove Global Constant Popup*

The Remove Global Constant menu is enabled if this attribute has a simple data type and was previously set with a global constant value.

2. Select **Remove Global Constant**. The global constant value associated with this attribute is removed.

## Set Local Constant

A local constant applies the attribute of a local instance ISO data type if the attribute has a simple data type. This instance of ISO data type can be used in modeling business objects or defining other ISO data types. The constant value applies when the attribute is not mapped to a table. Perform the following steps to set a local constant with an attribute of a local instance of ISO data type:

1. Right-click the attribute node in the object model. A popup menu appears.

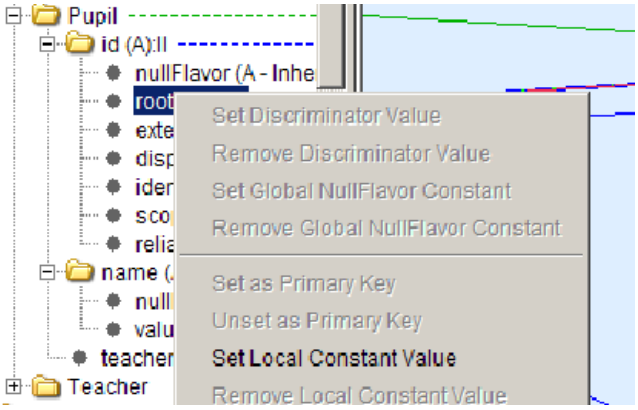



*Figure 2.18 Set Local Constant Popup*

The Set Local Constant menu is enabled if this attribute has a simple data type.

2. Select **Set Local Constant**. The Define Constant Value dialog box appears.

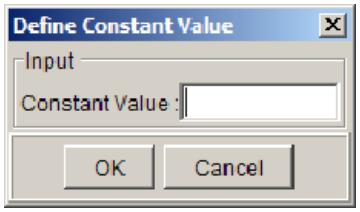



*Figure 2.19 Set Local Constant Dialog Box*

3. Enter a value and then click **OK**. A new local constant is added to the selected attribute.

## Remove Local Constant

1. Right-click an attribute node in the object model. A popup menu appears.

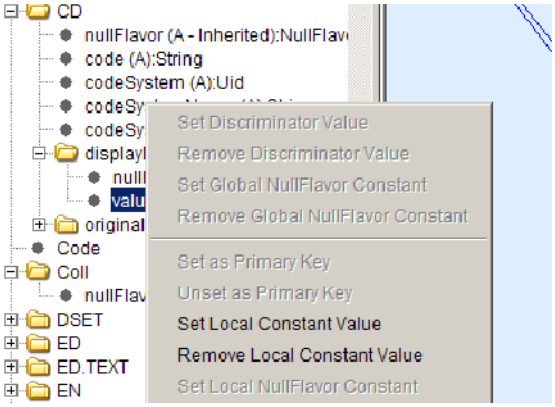



*Figure 2.20 Remove Local Constant Popup*

The Remove Local Constant menu is enabled if this attribute has a simple data type and was previously set with a local constant value.

2. Select **Remove Local Constant**. The local constant value of the selected attribute is removed.

## Set Global NullFlavor Constant

Global NullFlavor constant applies to all ISO data types. Perform the following steps to set a global NullFlavor constant with an ISO data type object.

1. Right-click an object node in the model of ISO data types. A popup menu appears.

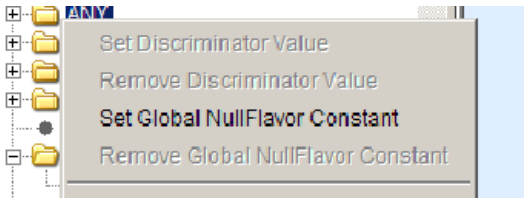



*Figure 2.21 Set Global Constant Popup*

The Set Global NullFlavor Constant menu is enabled.

2. Select **Set Global NullFlavor Constant**. The Define Global NullFlavor Constant Value dialog box appears.

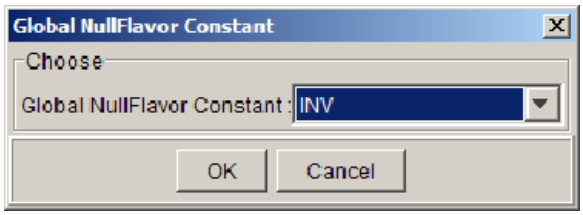



*Figure 2.22 Set Global NullFlavor Constant Dialog Box*

3. Choose a value from the list and then click **OK**. A new global NullFlavor constant is added to the selected ISO data type object.

## Remove Global NullFlavor Constant

1. Right-click an object node in the model of ISO data types. A popup menu appears.


*Figure 2.23 Remove Global NullFlavor Constant Popup*

The Remove Global NullFlavor Constant menu is enabled if a global NullFlavor constant was previously set to the selected object ISO data type.

2. Select **Remove Global NullFlavor Constant**. The global NullFlavor constant value is removed from the selected ISO data type object.

## Set Local NullFlavor Constant

Local NullFlavor constant applies to any instance of all ISO data types. This instance can be used in either modeling business objects or defining an ISO data type object. Perform the following steps to set a local NullFlavor constant with the instance of an ISO data type.

1. Right-click an object node in the object model or attribute with complex ISO data type. A popup menu appears.

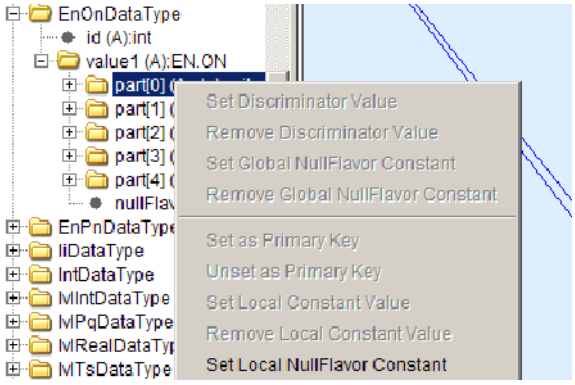



*Figure 2.24 Set Local NullFlavor Constant Popup*

The Set Local NullFlavor Constant menu is enabled.

2. Select **Set Local NullFlavor Constant**. The Local NullFlavor Constant Value dialog box appears.

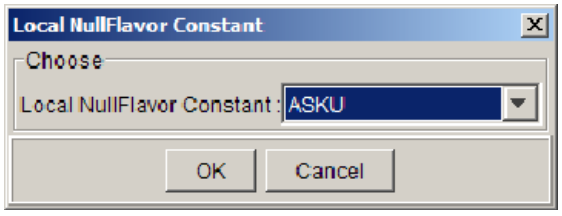



*Figure 2.25 Set Local NullFlavor Constant Dialog Box*

3. Choose a value from the list and then click **OK**. A new local NullFlavor constant is added to the selected instance of ISO data type.

## Remove Local NullFlavor Constant

1. Right-click an object node in the object model or attribute with a complex ISO data type. A popup menu appears.

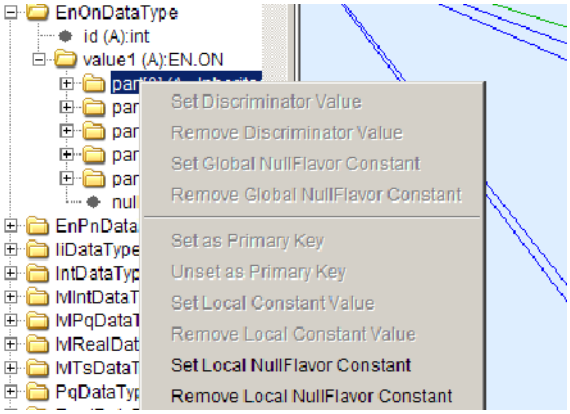



*Figure 2.26 Remove Local NullFlavor Constant Popup*

The Remove Local NullFlavor Constant menu is enabled if a local NullFlavor constant was previously set to the selected instance of an ISO data type.

2. Select **Remove Local NullFlavor Constant**. The Local NullFlavor constant value is removed from the selected instance of an ISO data type.

## Map Complex Data Type with Collection Attribute

This scenario maps complex ISO data types that have an attribute that is a collection of other ISO data types. The elements in the collection attribute are recursively mapped to columns of the same table that are mapped with the parent class. Perform the following steps to map a complex data type with a collection attribute.

1. Create a dependency mapping between the object in the object model panel and its target table in the data model panel. The object has one or more attributes with an ISO complex data type and a collection attribute. For example, in *Figure 2.27*, a dependency mapping is created between object “EnDataType” and table “EN_DATATYPE”.

2. Map all attributes with a simple ISO data type or a complex ISO data type without collection attribute. For example, in *Figure 2.27*, “EnDataType.id” attribute has the primitive data type “int” and it is mapped to column “EN_DATATYPE.ID” .

3. Click and drag the nested leaf attribute of the element of the collection attribute in the object model panel to the table column in the data model panel. For example, in *Figure 2.27*, the nested attribute “EnDataType.value.part[0].value is mapped to column “EN_DATATYPE.VALUE1_PN_VALUE”.

4. Repeat step 3 to map any other nested attribute of this element of collection attribute.

5. Repeat step 3 and step 4 to map the nested attributes of all other elements of the collection attribute. For example, in *Figure 2.27*, the attributes of the first and second element of the collection attribute are mapped to the target table.

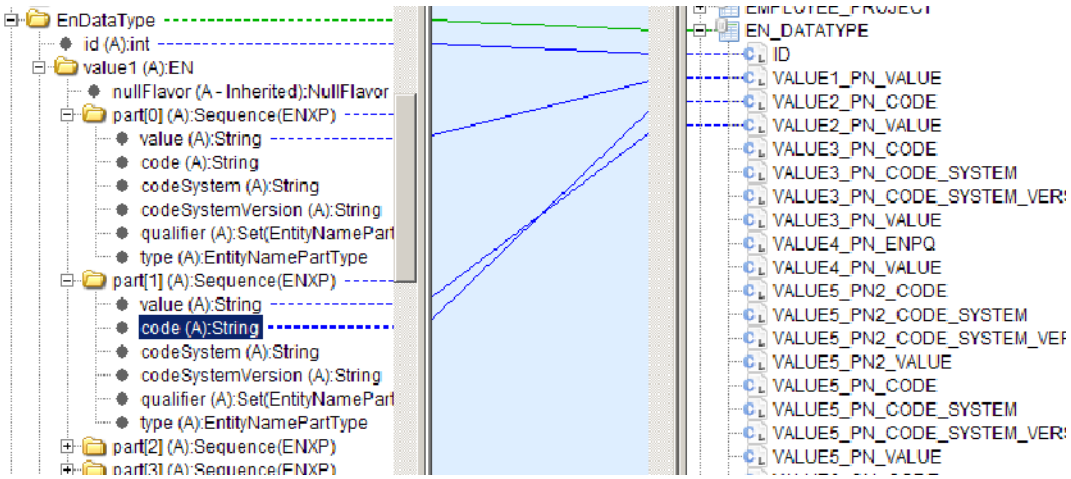



*Figure 2.27 Map Complex Data Type with Collection Attribute*

## Map Collection Data Types without a Joint Table

Collection data types contains a collection of other complex data type. The collection element can be either a simple or complex ISO data type. In this mapping scenario, the structures of all these collection elements are recursively mapped to columns of the same table. Perform the following steps to map elements of the collection data type.

1. Create a dependency mapping between the object in the object model panel and its target table in the data model panel. The object has an attribute with the ISO collection data type. For example, in *Figure 2.28*, a dependency mapping is created between object "DsetTelDataType" and table "TEL_DSET_DATATYPE".
2. Map all attributes that have a simple ISO data type or a non-collection complex ISO data type. For example, in *Figure 2.28*, "DsetTelDataType.id" attribute has the primitive data type "int" and it is mapped to column "TEL_DSET_DATATYPE.ID".
3. Click and drag the nested leaf attribute of the element of collection attribute in the object model panel to the table column in the data model panel. For example, in *Figure 2.28*, the nested attribute "DsetTelDataType.value1.item.value" is mapped to column "TEL_DSET_DATATYPE.VALUE1_VALUE".
4. Repeat step 3 to map any other nested attribute of this element of collection data type.
5. Repeat step 3 and step 4 to map the nested attributes of all other elements of the collection data type. For example, in *Figure 2.28*, the attributes of all three elements of the collection data type are mapped to the target table.

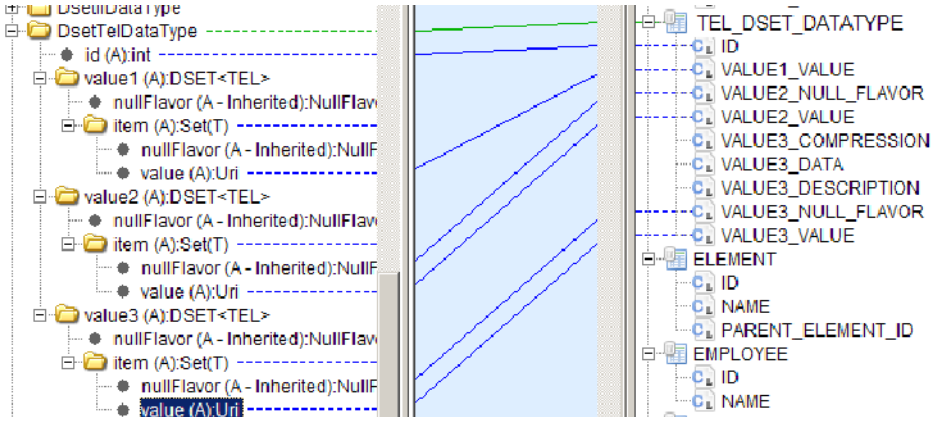



*Figure 2.28 Map Collection Complex Data Type Without Joint Table*

## Map Collection Data Types with a Joint Table

In this mapping scenario, the parent class and other non-collection attributes are mapped to the same target table. But the elements' collection data types are mapped to separate tables. Perform the following steps to map elements of the collection data type.

1. Create a dependency mapping between the object in the object model panel and its target table in the data model panel. The object has one or more attributes with the collection complex data type, which is a collection of other data types. For example, in *Figure 2.29*, a dependency mapping is created between object "DsetIiDataType" and table "DSET_II".
2. Map the non-collection attribute to the same target table as the parent class. For example, in *Figure 2.29*, "DsetIiDataType.id" attribute has the primitive data type "int" and it is mapped to column "DSET_II.ID" .
3. Click and drag the element of an attribute with the collection data type in the object model panel to the primary key column of a separate table in data model panel. For example, in *Figure 2.29*, the nested attribute "DsetIiDataType.value1" is mapped to column "DSET_II_VALUE1.DSET_II_ID".
4. Click and drag the nested attribute of the element in object model panel to the column of the table mapped with the element in data model panel. For example, in *Figure 2.29*, the nested attribute "DsetIiDataType.value1.extension" is mapped to column "DSET_II_VALUE1.DSET_II_EXTENSION".
5. Repeat step 4 to map any other nested attribute of this element of collection data type.
6. Repeat step 3, step 4, and step 5 to map all other elements of the collection data type. For example, in *Figure 2.29*, the two elements of the collection are mapped to two different target tables.

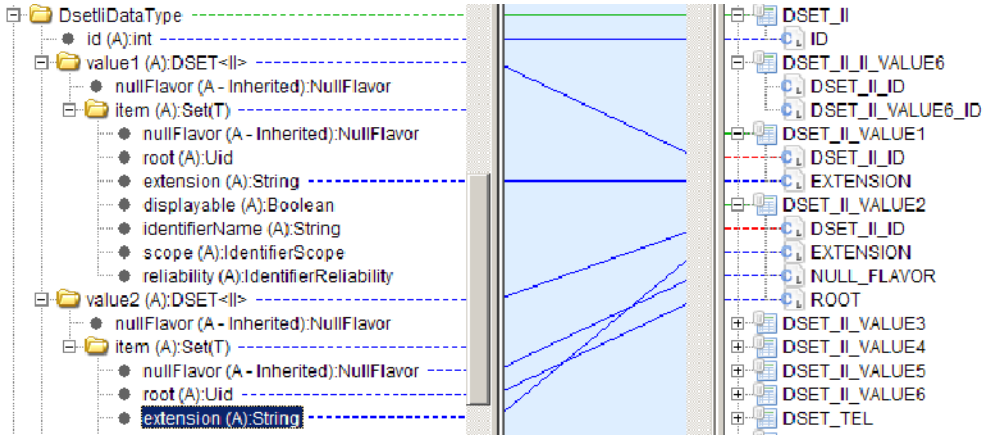



*Figure 2.29 Map Collection Data Type with Joint Table*

## Map Collection Data Type with a Collection Attribute

In this mapping scenario, the domain object has one or more attributes with the collection data type. The element of the collection data type is an ISO data type with a nested collection attribute of another ISO data type. The parent class and other non-collection attributes are mapped to the same target table. But these elements of the collection data type that refer to a data type with a collection nested attribute are mapped to separate tables. Perform the following steps to map elements of the collection data type with the collection attribute.

1. Create a dependency mapping between the object in the object model panel and its target table in the data model panel. For example, in *Figure 2.30*, a dependency mapping is created between object "DsetAdDataType" and table "DSET_AD_DATATYPE".
2. Map the non-collection data type attribute to the same target table of the parent class. For example, in *Figure 2.30*, "DsetAdDataType.id" attribute has the primitive data type "int" and it is mapped to column "DSET_AD_DATATYPE.ID".
3. Click and drag the element of the collection data type in the object model panel to the primary key column of a separate table in data model panel. For example, in *Figure 2.30*, the nested attribute "DsetAdDataType.value4" is mapped to column "DSET_AD_VALUE4.DSET_AD_DATATYPE_ID".
4. Click and drag the nested attribute of the selected in step 3 to the column of the table mapped with the element in data model panel. For example, in *Figure 2.30*, the nested attribute "DsetAdDataType.value4.item.part[0].code" is mapped to column "DSET_AD_VALUE4.ADXP_AL1_CODE".
5. Repeat step 4 to map any other nested attribute of the element of collection data type. For example in *Figure 2.30*, all the nested attribute of the collection attribute are mapped to the same table "DSET_AD_VALUE4".
6. Repeat step 3, step 4, and step 5 to map all other elements of the collection data type.

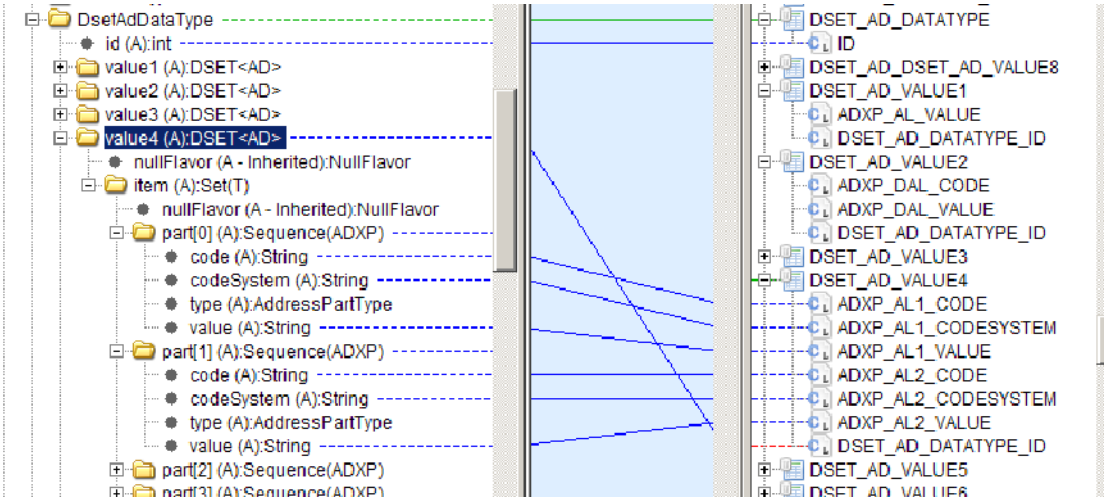



*Figure 2.30 Map Collection Data Type with Collection Attribute*

## Set Collection Element Type

The complex data types with collection attribute are used in two mapping scenarios: *Map Complex Data Type with Collection Attribute*, and *Map Collection Data Type with Collection Attribute*. In these mapping scenarios, all elements of the collection attribute have the same ISO data type structure. but each element only partially represents the whole data. For the example of collection attribute Sequence<ADXP>, each element has the data type ADXP. In order to distinguish these elements of the same collection attribute, they are annotated with tag of “mapped-collection-element-type”. Perform the following steps to set the Collection Element Type for the element of collection attribute.

1. Right-click an element of a collection attribute in the model. A popup menu appears.

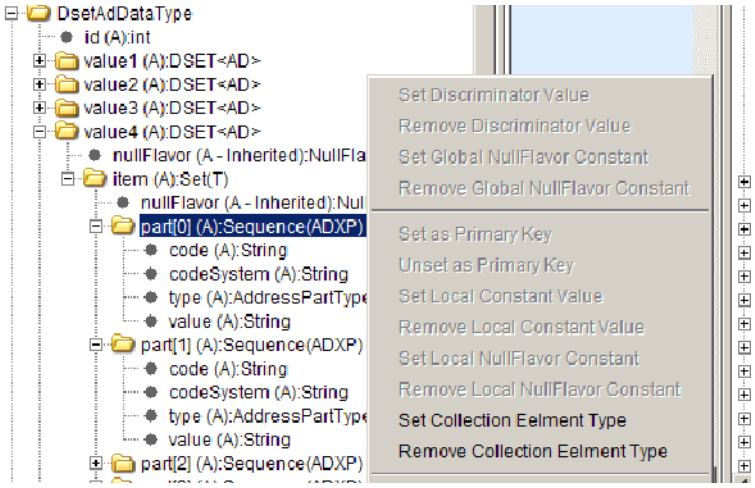



*Figure 2.31 Set Collection Element Type Popup*

The Set Collection Element Type menu is enabled.

2. Select **Set Collection Element Type**. The Collection Element Type dialog box appears.

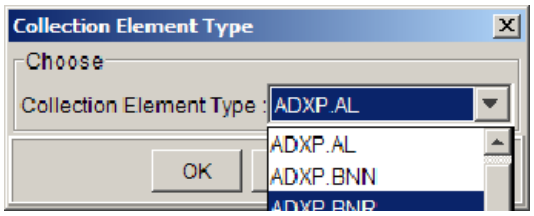



*Figure 2.32 Select Collection Element Type Dialog Box*

3. Choose a value from the list and then click **OK**. A new Collection Element Type value is added to the selected element of the collection attribute.

## Remove Collection Element Type

Perform the following steps to set Collection Element Type for the element of collection attribute.

1. Right-click an element of a collection attribute in the model. A popup menu appears.

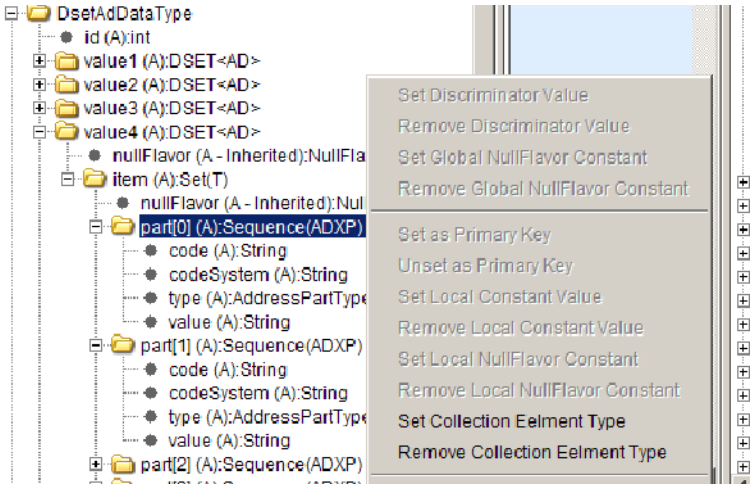



*Figure 2.33 Remove Collection Element Type Popup*

The Remove Collection Element Type menu is enabled if the selected element of collection attribute has been previously set with value of Collection Element Type.

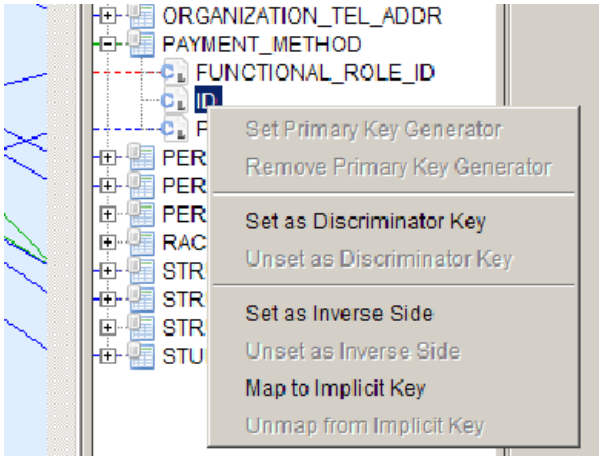
2. Select **Remove Collection Element Type**. The value of Collection Element Type is removed from the selected element of the collection attribute.

## Map to Implicit Key

When a complex data type is mapped to a table other than its parent business object, the primary key column of the target table must map to the key attribute of the source object node. The key attribute of the source object node can either be explicit or implicit. If the source node is not an explicit key attribute, the primary key column

should map to the implicit key attribute. Perform the following steps to map a primary key column to the implicit key attribute of a source object node.

1. Right-click an unmapped primary key column of the target table. A popup appears.




   The Map to Implicit Key menu option is enabled.

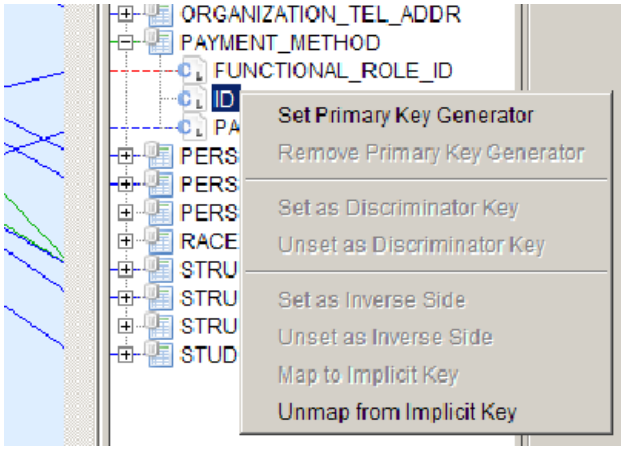
2. Select **Map to Implicit Key**. The primary key column is now mapped to the implicit key attribute of the source node.

   **Note:** Only one column can be mapped to the implicit key attribute of a source node.

## Unmap From Implicit Key

Perform the following steps to unmap the primary key column from the implicit key attribute.

1. Right-click the primary key column of the target table. A popup appears.



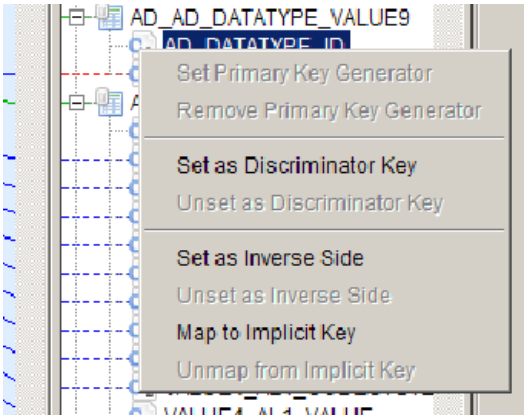

   The Unmap from Implicit Key menu option is enabled if the column has been previously mapped to the implicit key attribute of the source node.

2. Select **Unmap from Implicit Key**. The selected primary key column is no longer mapped to the implicit key attribute of the source node.

## Set as Inverse Side

In the Map Collection Data Type with Joint Table scenario, a correlation is created once a column of the joint table is mapped to a collection element node in the object model. Other unmapped columns of the joint table can then be set as the inverse side of the correlation. Perform the following steps to set a column as the inverse side

1. Right-click an unmapped column node of a joint table. A popup menu appears.




   The Set as Inverse Side menu option is enabled.

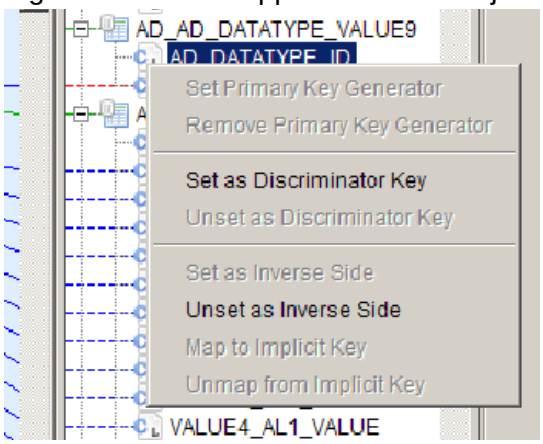
2. Select **Set as Inverse Side**. The selected column is now the inverse side of correlation between collection data type and its element.

   **Note:** Only one column can be set as inverse side of a correlation.

## Unset as Inverse Side

Perform the following steps to unset a column of joint table as the inverse side.

1. Right-click an unmapped column of a joint table. A popup appears.




   The Unset as Inverse Side menu option is enabled if this column has been previously set as the inverse side.

2. Select **Unset as Inverse Side**. The selected column is now no longer the inverse side of the correlation between a collection data type and its element.

# Understanding 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 one 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 needs to be mapped; the other association end does not need to be mapped.

In the example in *Figure 2.34*(Protocol and Amendment), 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`).

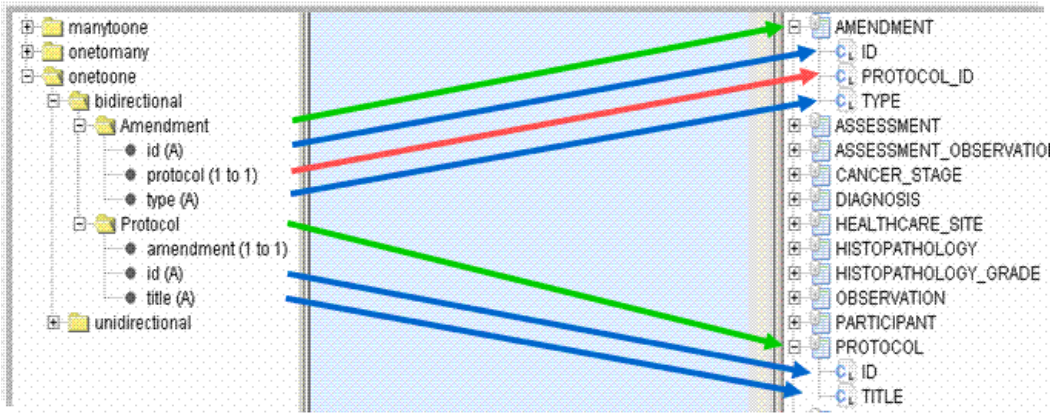



*Figure 2.34 One-to-One Bi-Directional Mapping*

## 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 2.35*, drag the association end (`HealthcareSite.address(1 to 1)`) and drop it onto the foreign key (`ADDRESS_ID`) of its persistence table (`HEALTHCARE_SITE`).

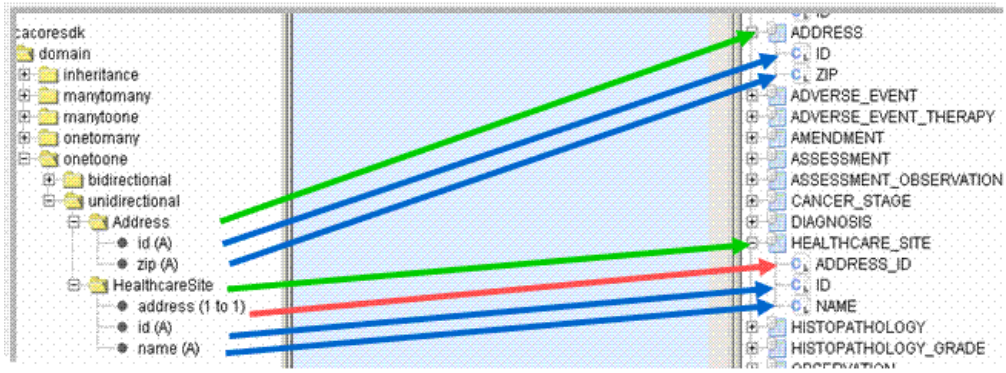



*Figure 2.35 One-to-One Uni-Directional Mapping*

## 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 2.36*, 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`).

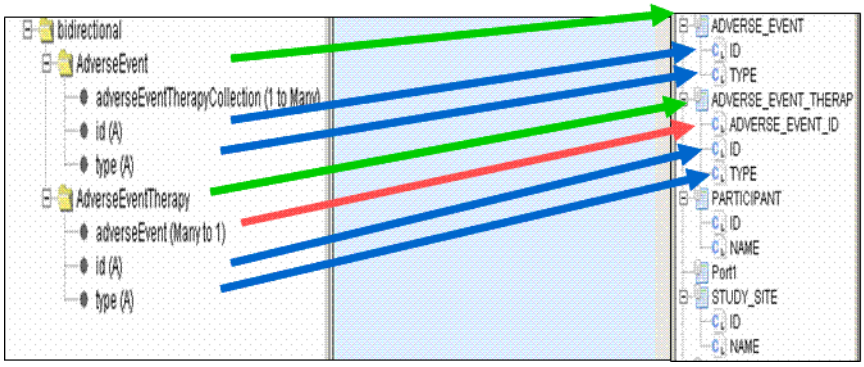



*Figure 2.36 One-to-Many Bi-Directional Mapping*

## 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 “many” side association end must have the foreign key column of the persistence table for the invisible “one” side association end object.
- Map the visible “many” side association end to the foreign key column of its persistence table.

In the example (`Histopathology` and `HistopathologyGrade`) in *Figure 2.37*, drag the association (`HistopathologyGrade.histopathology(Many to 1)`) and drop it onto the foreign key (`HISTOPATHOLOGY_ID`) of the corresponding table (`HISTOPATHOLOGY_GRADE`).

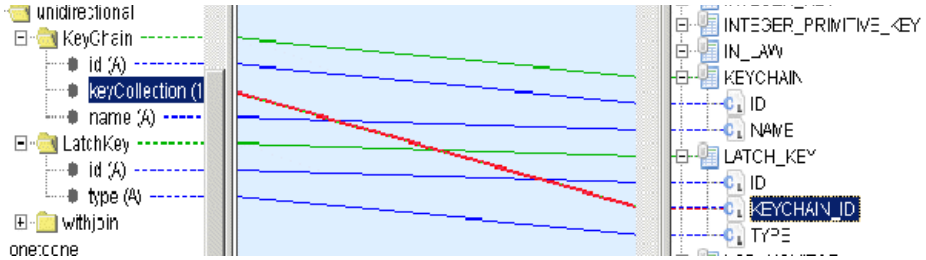



*Figure 2.37 Many-to-One Uni-Directional Mapping*

## 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`.

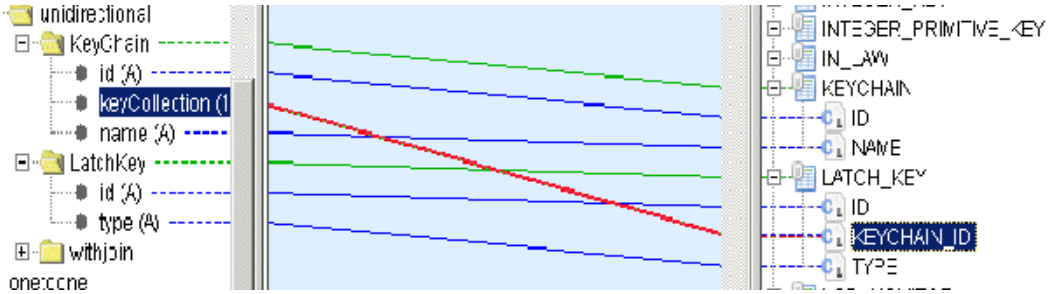



*Figure 2.38 One-to-Many Uni-Directional Mapping*

## Many-to-Many Bi-Directional

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

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

In the example in *Figure 2.39*, 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.

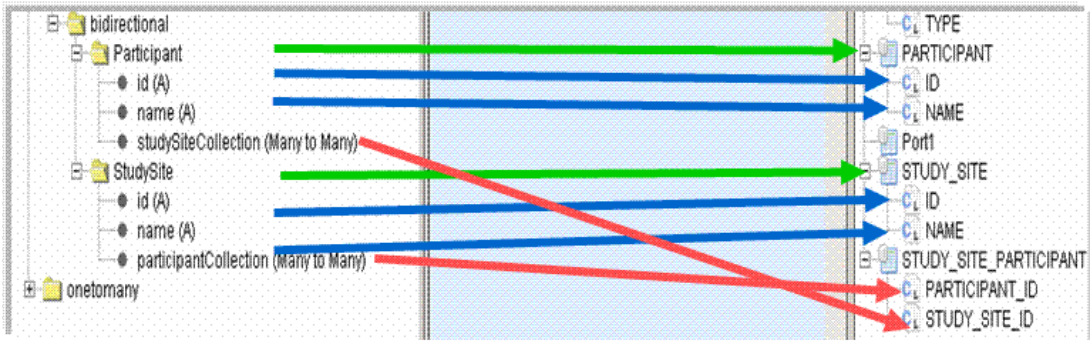



*Figure 2.39 Many-to-Many Bi-Directional Mapping*

## 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 columns of the persistence tables for both association end objects.
- Map only the visible association end to the foreign key column of the "correlation-table".

In the example in *Figure 2.40*, 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`).

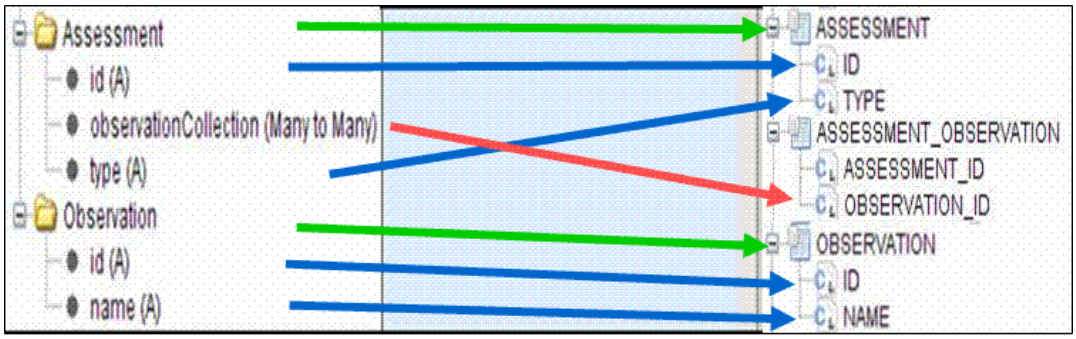



*Figure 2.40 Many-to-Many Uni-Directional Mapping*

## Understanding 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 difficult to map class hierarchies to a relational database model. caAdapter Model Mapping Service supports the following three basic Hibernate mapping strategies:

- Table per class (see *Table Per Class Hierarchy* on page 42)
- Table per subclass (see *Table Per Subclass* on page 43)
- Table per concrete class (see *Table Per Concrete Class* on page 45)

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

**Note:** *Implicit Polymorphism* means that instances of the class will be returned by a 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.

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

In the following example, all classes in the hierarchy are persisted with the same table `SHOES`. The class `Shoe` is the super class with two attribute: `id` and `color`. The attribute `Shoe.id` is mapped to column `SHOES.ID`, and the 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 attributes, `id` and `color`, but they are not mapped. The subclass `DesignerShoes` has the association `designer` to object `Designer`, which is mapped to the foreign key column `SHOES.DESIGNER_ID`. The subclass `SportShoes` has an additional attribute `sportType`, which is mapped to the column `SHOES.SPORT_TYPE`.

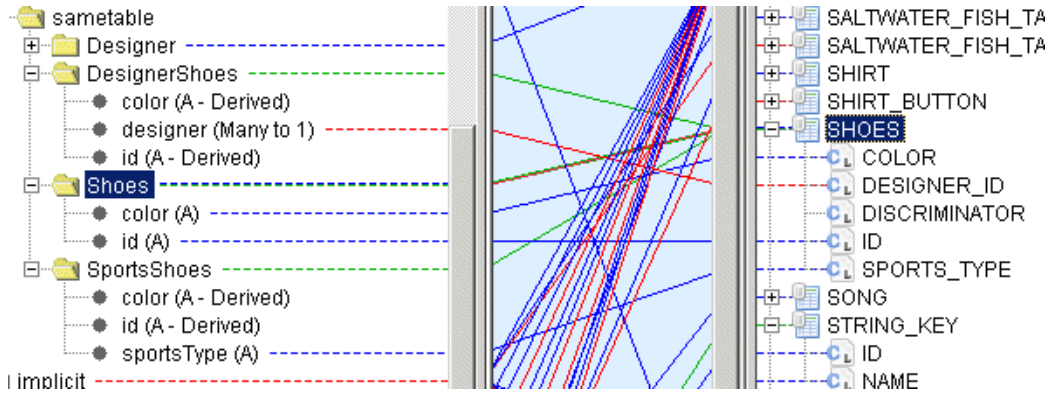



*Figure 2.41 Mapping of a Table from the Whole Class Hierarchy*

## Table Per Subclass

This strategy uses a different table for each class (super class and all subclasses) in the hierarchy. Each class maps its attributes to its own persistence table, as well as all of its associations to the associated class/object. 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 into the database. It also uses the same primary key to perform a "JOIN" operation when it accesses the database.

The following mapping rules apply to Table Per Subclass:

- Each class maps to the individual persistence table.
- All tables 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 separate tables. The class `Payment` is the super class with two attributes: `id` and `amount`. The attribute `Payment.id` is mapped to the column `PAYMENT.ID` and the attribute `Payment.amount` is mapped to column `PAYMENT.AMOUNT`. The subclasses do not map the inherited attribute `amount` but rather map the inherited attribute `id` since it is the primary key. The subclass `Credit` maps its additional attribute `Credit.cardNumber` to the column `CREDIT.CARD_NUMBER`, and its association `Credit.issuingBank` to the foreign key column `CREDIT.BANK_ID`.

The following is an example of the mapping of the super class.

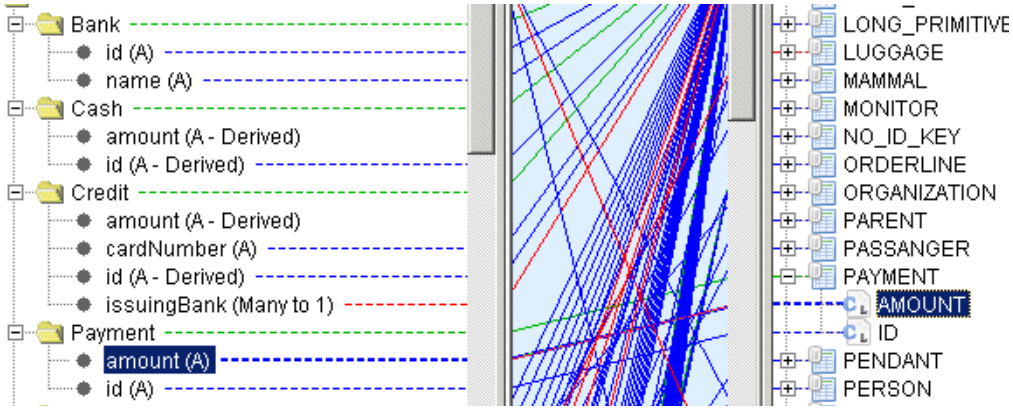



*Figure 2.42 Mapping of the Super Class*

The following is an example of the mapping of hte subclass with the inherent attribute as the primary key.

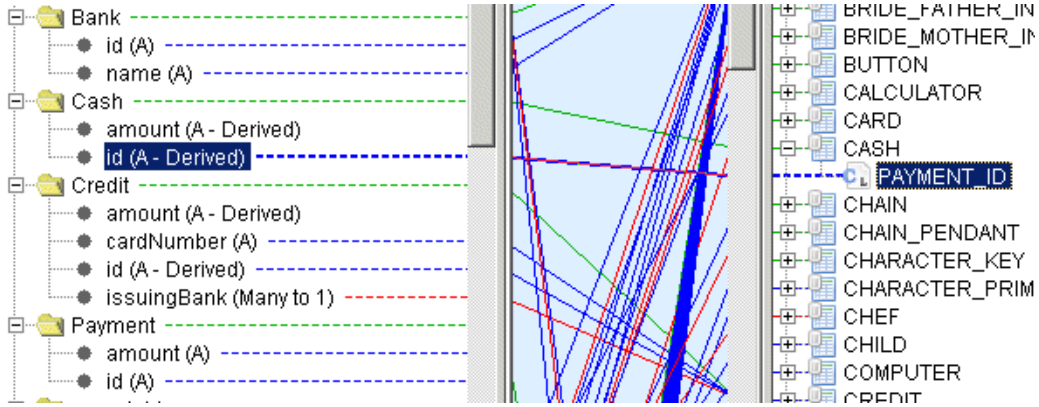



*Figure 2.43 Mapping of the Subclass with the Inherent Attribute as the Primary Key*

The following is an example of the mapping of the subclass with an additional attribute.

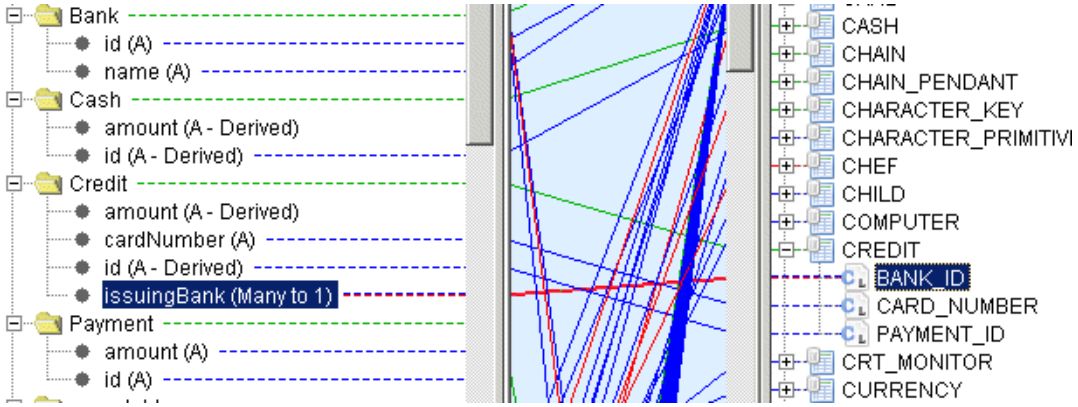



*Figure 2.44 Mapping of the Subclass with an Additional Attribute*

## 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 to be 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:

- Do not map super class to any persistence table.
- Map each subclass to its own persistence table.
- All these tables share the same primary key.
- Map all attributes (including inherited attribute) of a subclass to its 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`.

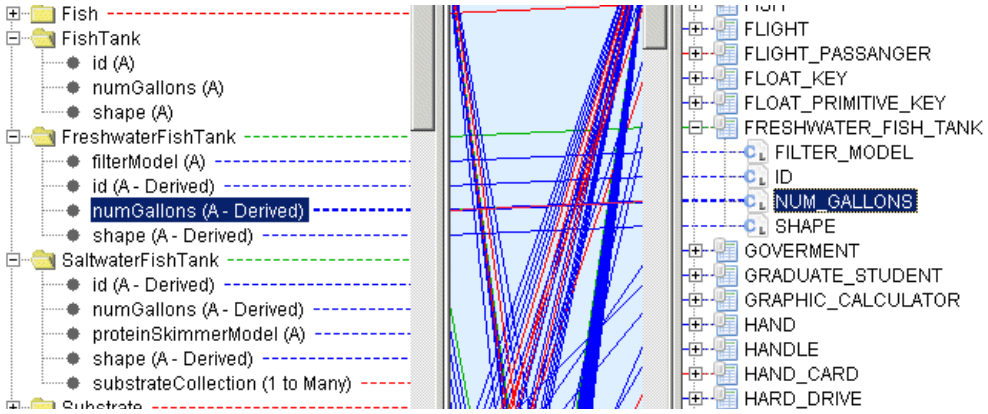



*Figure 2.45 Mapping of the Concrete Class*

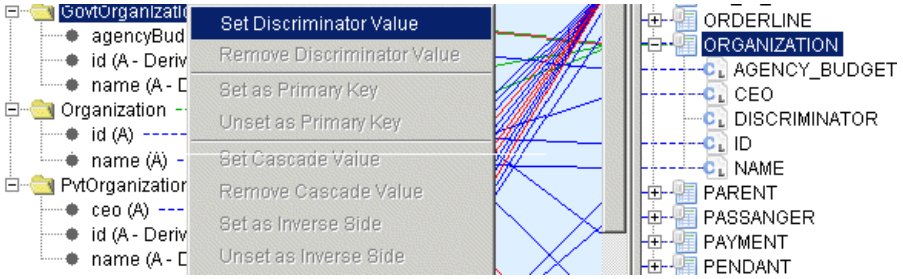
# Additional Operations on Object Model

## 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 with 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 subclass.

### Set a New Discriminator Value

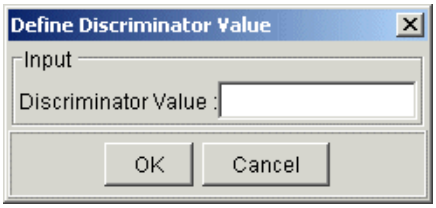
1. Right-click an object in the left panel. A popup window appears.




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.



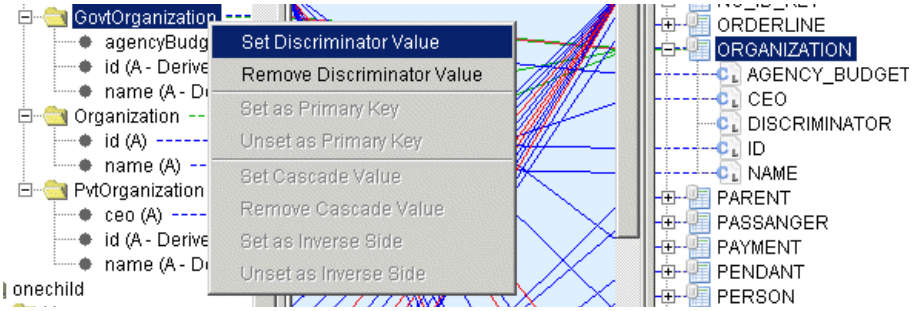

3. Enter a value in the Discriminator Value box.

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

4. Click **OK**. The value you entered is assigned to the selected class as the discriminator value.

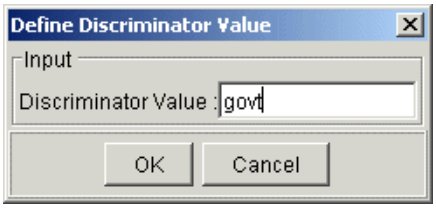
## Change a Discriminator Value

1. Right-click 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.

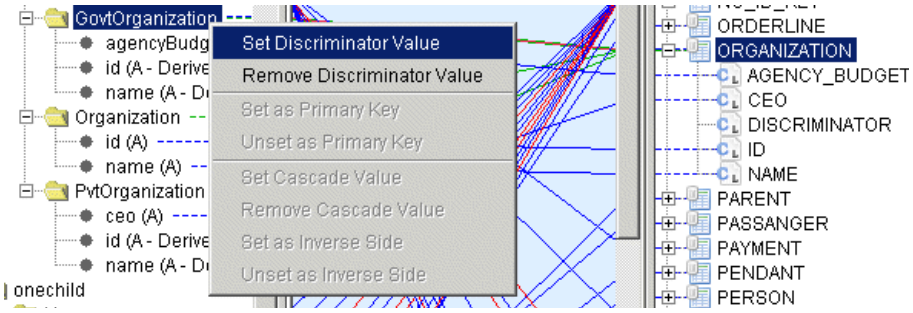



3. In the Discriminator Value box, enter a new value.

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 object in the left panel. A popup window appears.



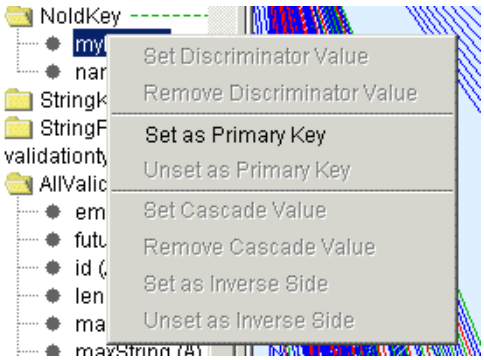

2. Select **Remove Discriminator Value**. The discriminator value is no longer assigned to the selected class.

## Primary Key Attribute

caCORE SDK requires you to mark an attribute 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.




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

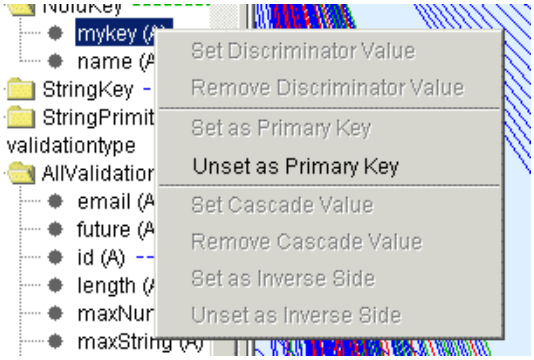
- the selected class attribute is mapped to a table column.
- no attribute is set as the class `identifier` attribute for the selected class

2. Select **Set as Primary Key**. The attribute you selected 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**.

   **Note:** 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

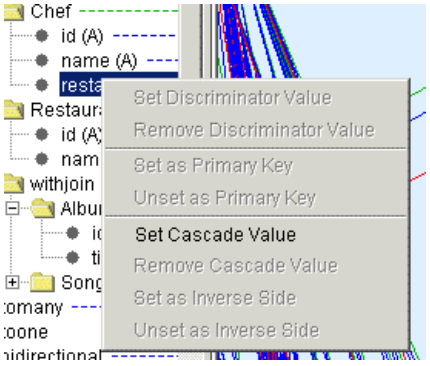
In Hibernate object-relational mapping, the cascade setting specifies which operations should cascade from the parent object to the associated object. Hibernate supports the following cascade setting values:

- 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 setting value, the SDK generates a Hibernate file without a cascade setting value. 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 Setting Value

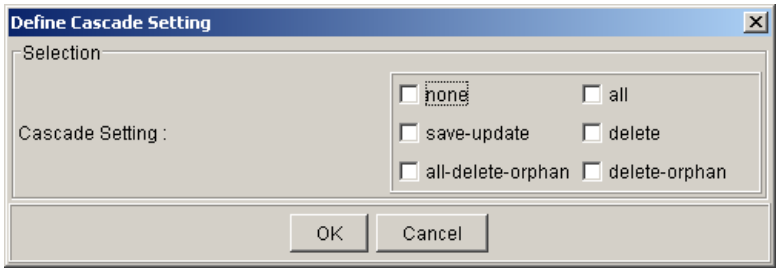
1. Right-click an association end on a class in the left panel. A popup window appears.




The Set 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

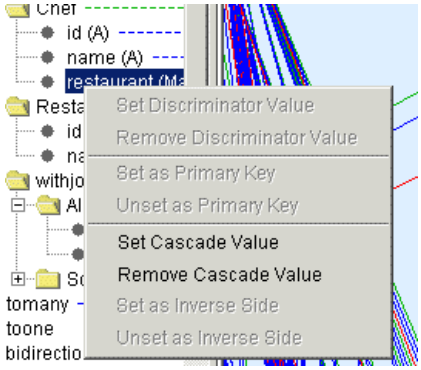
2. Select **Set Cascade Value**. The Define Cascade Setting dialog box appears, showing all permissible values as check boxes.




3. Select the check box with the preferred value.
4. Click **OK**. The selected value is now the class cascade setting value for the selected association end.

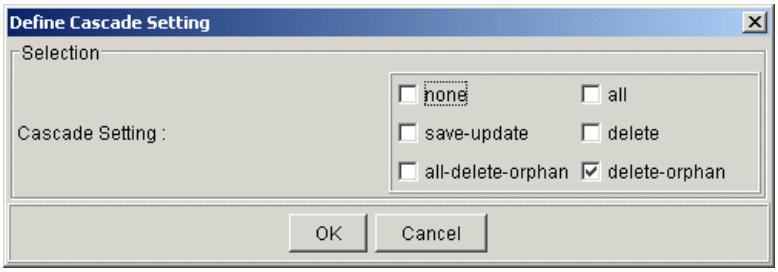
## Update an Existing Cascade Setting

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




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

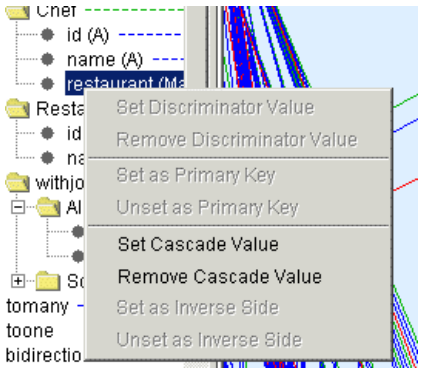
2. Select **Set Cascade Value**. The Define Cascade Setting dialog box appears showing the existing cascade style value.




3. Select a new cascade setting value.
4. Click **OK**.

## Unset Cascade Value

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



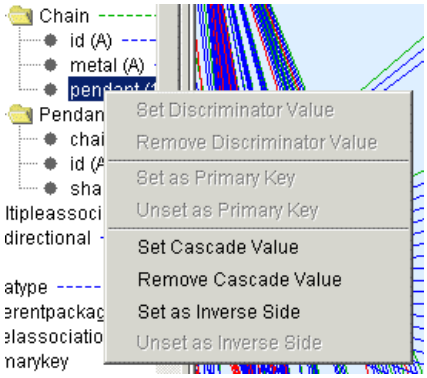
2. Click the **Remove Cascade Value** menu. The cascade style value assigned to association end is removed.

## 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 association end on a class in the left panel. A popup menu appears.




The Set as Inverse Side menu option is enabled if

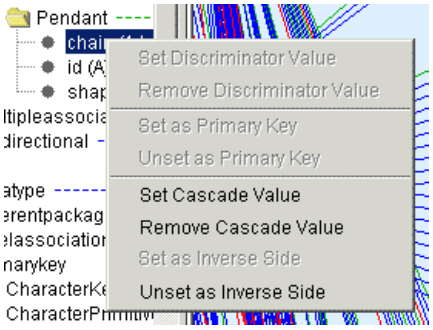
- the selected association end is mapped to a table column
- correlation table is used in the association mapping
- the selected association end was not previously set as inverse side

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 set one side as 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.




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

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

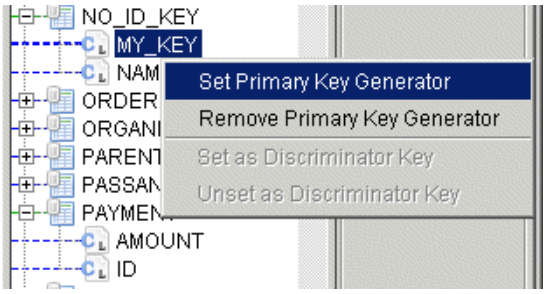
## Additional Operations on Data Model

### 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 an individual class.

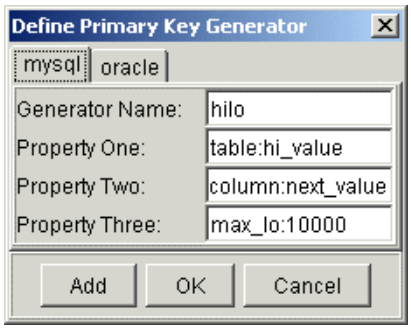
### Define a New Primary Key Generator

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

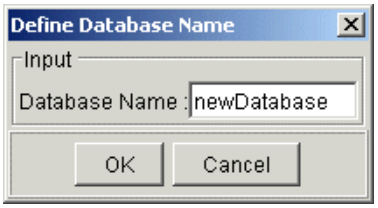



The Set Primary Key Generator menu is enabled if this column is a primary key column.

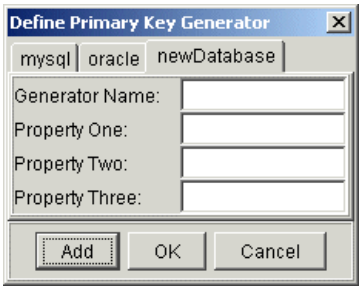
2. Select **Set Primary Key Generator**. The Define Primary Key Generator window appears.




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



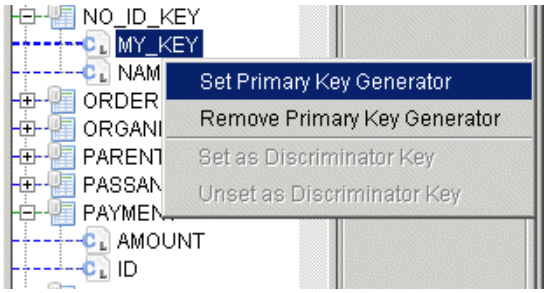

4. Type the name of the database and click **OK**. A new tab is added for the new database with fields where you can enter required properties.




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

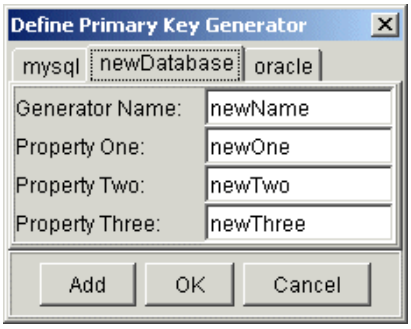
## Update a Primary Key Generator

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.

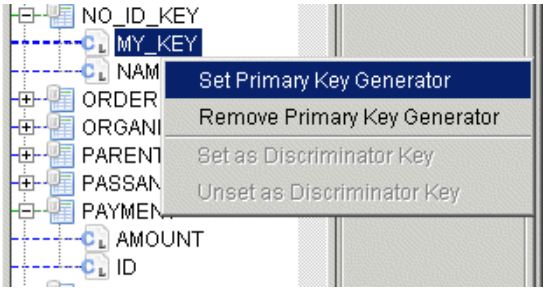
2. Select **Set Primary Key Generator**. The Define Primary Key Generator window appears.
3. Select the primary key generator tab.




4. Change the value of any property.
5. 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.




**Note:** 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.

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

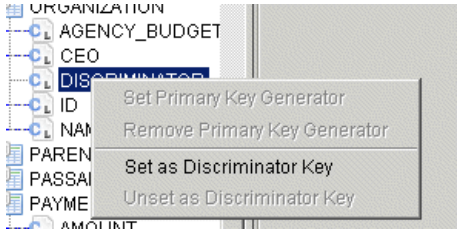

3. Select the name of a primary key generator from the list.
4. Click **OK**. The selected primary key generator is removed from the selected table.

# 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 the value of 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

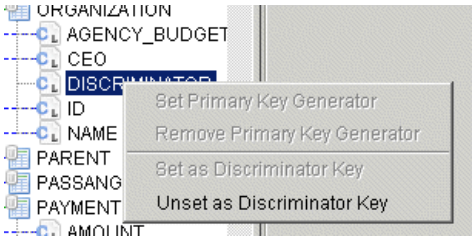
- the table is mapped using the Table Per Hierarchy strategy
- the select column is not mapped to any `object.attribute`

2. Select **Set as Discriminator Key**. The selected column is now the table's discriminator column.

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

## 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 column and its is available to map from any object.attribute.

# User Interface Legend

See the following sections for a key to abbreviations and mapping line colors in caAdapter MMS.

## 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

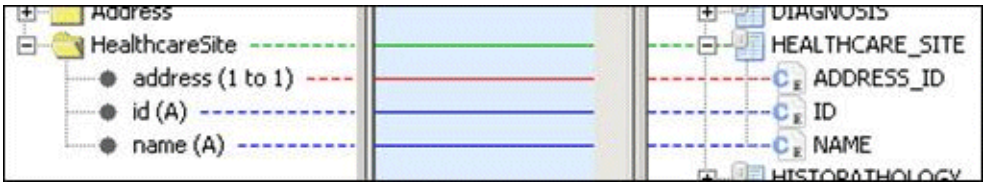



*Figure 2.46 Mapping Line Colors*

# Prefix for Object and Data Model

Specify the prefix to use for object or data model elements using the Tools > Preferences menu option (*Figure 2.47*).

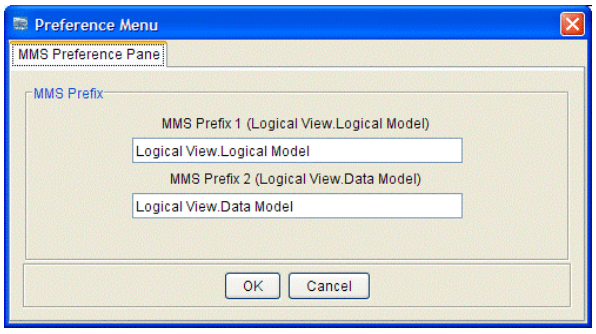



*Figure 2.47 Designating Prefixes on the Preference Menu*

## 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 caAdapter MMS 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


## A

Association mapping, creating 21

## C

caAdapter
- core engine architecture 6
- Model Mapping Service, about 13
- overview 5

caBIG solution 5
caCORE
- generate application using XMI file 18

caCORE SDK, definition 61
CLOB, definition 61

## D

Deleting mapping lines 22
Dependency mapping, creating 20

## E

EA, definition 61
Existing map specification, opening 17

## G

Generating XMI file 15

## H

HBM files 19
Hibernate mappings 19
HL7 7

## M

Mapping line colors 56
Mapping lines, deleting 22
Mapping scenario
- many-to-many bi-directional 40
- many-to-many uni-directional 41
- one-to-many/many-to-one bi-directional 38
- one-to-many/many-to-one uni-directional 39
- one-to-one bi-directional 37
- one-to-one uni-directional 37

Mapping specifications
- saving 18
- validating 17

Model Mapping Service Tool
- architecture 7
- creating object model to data model map specification 16
- functions 14
- generating an XMI file from EA 15
- integrating with other components 14
- opening an existing map specification 17
- using to generate caCORE SDK code 14

## N

Node details
- 1 to 1 56
- 1 to Many 56
- A 56
- A-Derived 56
- Many to 1 56
- Many to Many 56

## O

Object model to data model map specification 16
Object to database mapping specification 19

## S

Saving mapping specifications 18
Study Data Tabulation Model 9

## U

UML, definition 61

## V

Validating mapping specifications 17

## X

XMI, definition 61
XMI file
    generate caCORE-like application 18
XML, definition 61