**caAdapter 4.1**

**Quick Start Mapping Guide Guidee**

caAdapter Mapping Tool

CSV to HL7 v3

CSV or Database to RDS (SDTM)

Object Mode to Data Model



***June 30, 2008***

Center for Biomedical Informatics and Information Technology



***This is a U.S. Government Work***

### Change Record

| Date | Author(s) | Document Version | Change Reference (Major Changes) |
| --- | --- | --- | --- |
| 11/07/05 | Dan Grandquist | 1.2 | First Draft |
| 11/15/05 | Dan Grandquist | 1.2 | Changes: Ye Wu and Matt Giordano |
| 01/31/2006 | Scott Jiang | 1.2 |  |
| 11/15/2006 |  | 3.2 | Added sections for CSV to RDS (SDTM) and Object to Database mapping sections |
| 11/07/2007 | Charles Yaghmour | 4.0 | Added sections for CSV to RDS (SDTM) and Object to Database mapping sections |
| 6/26/2008 |  | 4.1 |  |

# Table of Contents

[Table of Contents 3](#_Toc182212330)

[1 Introduction 4](#_Toc182212331)

[2 CSV to HL7 v3 Mapping 4](#_Toc182212332)

[2.1 Introduction 4](#_Toc182212333)

[2.2 Hierarchical Tree Structure 5](#_Toc182212334)

[2.3 Structural Relationship 6](#_Toc182212335)

[2.4 Basic Mapping Structure 10](#_Toc182212336)

[2.5 Mapping Example 21](#_Toc182212337)

[2.6 Conclusion 25](#_Toc182212338)

[2.7 A Word about HL7 v2.x to HL7 v3 Mapping 25](#_Toc182212339)

[3 CSV to RDS (SDTM) Mapping 26](#_Toc182212340)

[3.1 Introduction 26](#_Toc182212341)

[3.2 Mapping Rules 27](#_Toc182212342)

[3.3 Mapping Example 27](#_Toc182212343)

[3.4 Recommendations for CSV Specification 28](#_Toc182212344)

[4 Model Mapping Service 28](#_Toc182212345)

[4.1 Introduction 28](#_Toc182212346)

[4.2 Basic Mapping 28](#_Toc182212347)

# Introduction

caAdapter provides a variety of mapping options with differing source and target formats. Each mapping type has its own business rules. The chapter on CSV to HL7 v3 Mapping serves two overall source formats as seen from the main menu of caAdapter: the CSV to HL7 v3 Mapping and Transformation Service and the HL7 v2 to HL7 v3 Mapping and Transformation Service. The following two chapters cover the CSV to SDTM Mapping and Object to Database Mapping respectively. This document describes the business rules and the nuances for how to map source to target fields and get the desired data results for each type of mapping.

# CSV to HL7 v3 Mapping

## Introduction

The caAdapter application contains four modules used in CSV to HL7 v3 mapping. The first two allow the user to define the specification for the source and target formats. The third is a mapping tool to help users map a comma-separated value (CSV) specification to an HL7 v3 message specification. The fourth is a transformation service to convert data from a source CSV data file format to a series of HL7 v3 messages, with the mapping file and a CSV data file as inputs.

The purpose of this chapter is to provide users a knowledge base for using the caAdapter Mapping Tool to map a comma-separated value (CSV) specification to an HL7 v3 message specification. This includes instructions on types of specification maps, kinds of specification mapping relationships and resulting output from specific mapping scenarios. To explain the mapping rules further, we will keep the result of the transformation service, that is, the series of HL7 v3 messages, in mind while discussing or illustrating the applications of various mapping rules.

For consistency, here we list a group of generally used terms that will be referenced in later context.

|  |  |
| --- | --- |
| Term | Explanation |
| CSV Specification | A comma-separated value (CSV) specification that presents the segment-based structure of a series of CSV data files |
| Source Specification | In current context, it refers to the CSV Specification |
| HL7 v3 Message Specification | An XML-based specification that is derived from the HL7 .mif file for the selected message type, with customized clone and attribute layout as well as other pre-defined values on data type fields |
| HL7 v3 XML Specification | Same as HL7 v3 Message Specification |
| Target Specification | In current context, it refers to the HL7 v3 Message Specification |
| Map Specification or Map File | An XML-based specification that records the mapping relationship between a source and target specification |
| CSV data file | A segment-based CSV data file that may conform to a certain CSV Specification hierarchically |
| Source data file | In current context, it refers to CSV data file |
| HL7 v3 Message | The outcome of the transformation service, i.e. the XML messages that comply to HL7 v3 specification |
| Target Message | In current context, it refers to HL7 v3 Message |

## Hierarchical Tree Structure

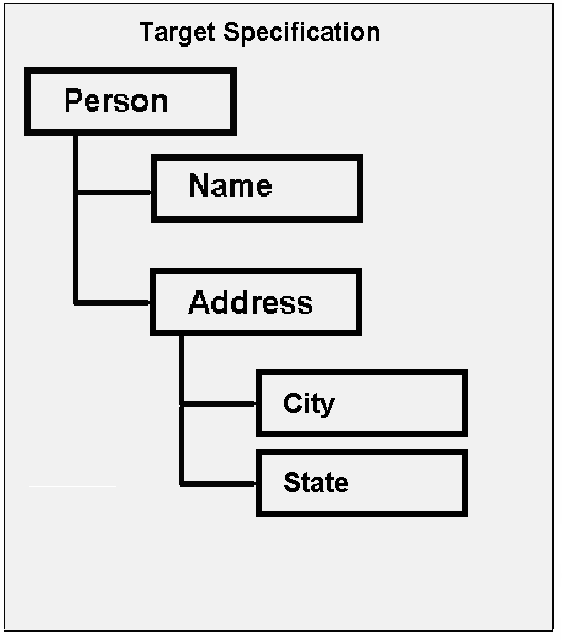


Figure Hierarchical Tree Structure

### Composite

A Composite is defined as a hierarchical structure that holds other hierarchical elements. As illustrated in , because the “Person” node holds the “Name” and “Address” nodes, it is a “Composite” structure. Similarly, because the “Address” node also holds the “City” and “State” nodes, it becomes a “Composite” structure as well. In general, a Composite could contain another Composite in its hierarchical structure.

### Leaf

A Leaf is defined as a hierarchical structure that does not hold other hierarchical elements. As illustrated in , because the “Name” node does not have any other hierarchical structure beneath it, it is a Leaf node. In general, a leaf node belongs to one and only one Composite structure, while a composite structure could hold more than one Leaf nodes, such as the “Address” Composite.

## Structural Relationship

### Parent-Child relationship

In a hierarchical tree structure, we refer to a node as a parent of another node if and only if the defined parent node hierarchically contains the other node. Similarly, we refer to the contained node is the child node of the aforementioned parent node.

For example, in the Figure below, under the source specification, the “Organization” node is the parent node of the “Doctor” node beneath it and the “Doctor” node is a child node of the “Organization” node.

In contrast, under the target specification, the “Doctor” node is the parent node of the “Organization” node beneath it.

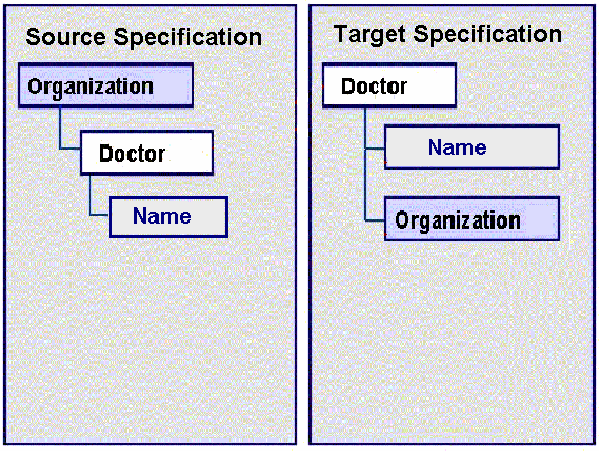


Figure Example of a parent child relationship

### Sibling Relationship

Segments, fields, clones, attributes, and data type fields are all types of elements. In source and target specifications, sibling elements are the structures that share an immediate common parent. For example, in the figure below, a “Doctor” node has “ID”, “Name” and “Address” as its child elements. The “ID”, “Name” and “Address” elements are collectively referred to as siblings under the context of the “Doctor” parent. In other words, a sibling relationship is always mentioned relative to a given parent element.

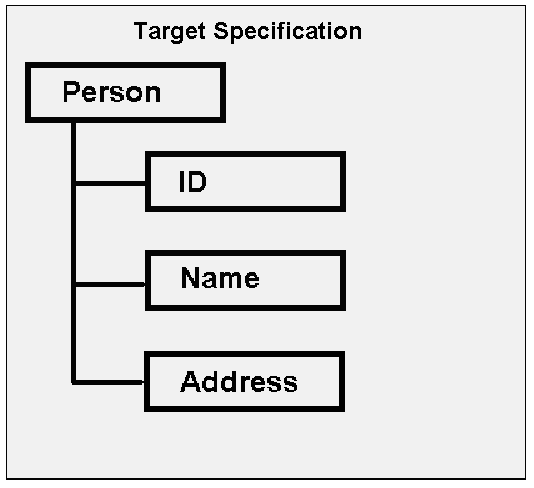


Figure Sibling Relationship

### Cardinality

Cardinality, also known as “Multiplicity”, defines the “*number of*” relationship between a parent and its child node, if any. For uniformity, we define the relationship from the parent node’s point of view, but record the cardinality information on the child node side in a properties attribute. Cardinality only denotes the multiplicity relationship between a given node and its immediate parent node.

In current release, cardinality is implied in the CSV source Specification, while it is explicit in the HL7 v3 message specification. We will elaborate on this further after defining different types of cardinality relationships we may encounter in the caAdapter application.

#### One-to-One

The one-to-one relationship, denoted as 1..1, implies that the one parent node only contains one and only one such child and conversely, one such child belongs to one and only one given parent node.

”, under the **target specification**, the “Name” node may have a one-to-one relationship with the parent “Doctor” node. In other words, if the underlying data contained one “Doctor” data record that included more than one “Name” data record, it would cause a validation error between the specification and the data file.

#### One-to-Many

The one-to-many relationship, as denoted as 1..\*, implies that the one parent node could contain one or more of such children and conversely, one or more of such children may belong to one given parent node.

”, under the **source specification**, the “Organization” node has a one-to-many relationship with the “Doctor” node. The definition implies that for one parent “Organization” node, it will contain minimally one “Doctor” node and may contain as many “Doctor” nodes as possible. In real life, it reflects the fact that a small clinic could have at least one doctor, while a large hospital may employ hundreds of doctors in different departments.

#### Cardinality in Source Specification

In the CSV source specification, cardinality is implied through the parent-child relationship between segments or between a segment and those fields underneath it. There are no numbers to enter or update for cardinality when defining a CSV specification. In fact, between fields and their parent segment, the cardinality is always one-to-one (1..1), that is, one field belongs to one and only one parent segment. From the aspect of parent and child segments, child segments are only needed when there may be a one-to-many relationship between data items. In other words, in the CSV source specification, the child segment has an implied one-to-many cardinality relationship to its parent segment. In the data file, child segments may appear zero, one or many times for each occurrence of a parent segment.

For instance, if the parent segment (PAT) contains data about a patient and the child segment (MED) contains data about medications the patient is taking, a CSV specification may look like this:

PAT

1: PatientId

2: LastName

3: FirstName

4: MiddleName

5: VisitDate

MED

1: AgentName

2: DoseAmount

3: DoseUOM

4: DoseFrequency

5: StartDate

And the data file may look like this:

PAT, 12345, Smith, John, Joseph, 20060123

MED, Ibuprofen, 200, mg, BID, 20060101

PAT, 23456, Green, Peter, David, 20060123

PAT, 34567, Parker, James, Robert, 20060123

MED, Ibuprofen, 200, mg, BID, 20060101

MED, Claritin, 100, mg, BID, 20051001

MED, Prevacid, 500, mg, BID, 20051201

This shows that Smith is taking one medication, Green is currently not taking any medications, and Parker is taking three.

#### Cardinality in Target Specification

Cardinality in an HL7 v3 message specification is inherited from the HL7 .mif file for the selected HL7 v3 message. Users of the caAdapter application cannot change this cardinality; they can only include or exclude optional portions of the message structure. For elements that are included in the target specification, the cardinality may be one-to-one or one-to-many.

## Basic Mapping Structure

### Mapping Types

Using the caAdapter Mapping Tool’s CSV to HL7 v3 Map Specification, a user creates links between source fields and target data type fields and between source segments and target attributes or clones. Links between source fields and target data type fields represent data relationships. Links between segments and clones or attributes explicitly relate conceptual groups of data to each other for the purpose of applying cardinality rules.

From such a perspective, we have

* *Data* *mappings* specifying the source of the *content* and the element in the target message where it will go, and
* *Concept* *mappings* controling the number of occurrences of message elements in the target messages generated

From a structural perspective, the data mapping is also referred to as the “leaf-to-leaf” mapping, that is, a leaf node in the source tree (source field) to a leaf node in the target tree (data type field).

The concept mapping, in contrast, is also referred to as the “composite-to-composite” mapping, that is, from a source segment to a target clone or attribute. The application does not allow the user, however, to create cross mappings between “composite” and “leaf” elements, that is, map a source field to a target clone or attribute, or map a source segment to a target data type field.

When the caAdapter application generates HL7 v3 messages, four factors will work together to govern the number of instances of message content to be rendered in the results. They are listed in order of consideration as follows:

1. The cardinality attribute of the given clone, attribute, or data type field;
2. Any explicit conceptual mapping to the given clone or attribute from the source segment;
3. If no explicit conceptual mapping is available, any implicit conceptual mapping to be derived by the given clone or attribute from the source segment, if an explicit mapping does not exist;
4. The data mapping to the given data type field from the source field;

The current caAdapter application will derive an implicit conceptual mapping from a source field’s or segment’s parent segment to a clone, attribute, or data type field’s parent clone or attribute, if there exists explicit data mappings between the aforementioned source segment or field and target clone, attribute, or data type field, and no explicit conceptual mapping activity is yet defined to reflect the cardinality relationship between the aforementioned source field’s or segment’s parent segment to the aforementioned clone, attribute, or data type field’s parent clone or attribute. For example, the mapping between a child segment and child clone will imply the existence of an implicit conceptual mapping between their respective parents, if no such mapping is explicitly created. In other words, factor 3 is only effectively derived in the absence of factor 2.

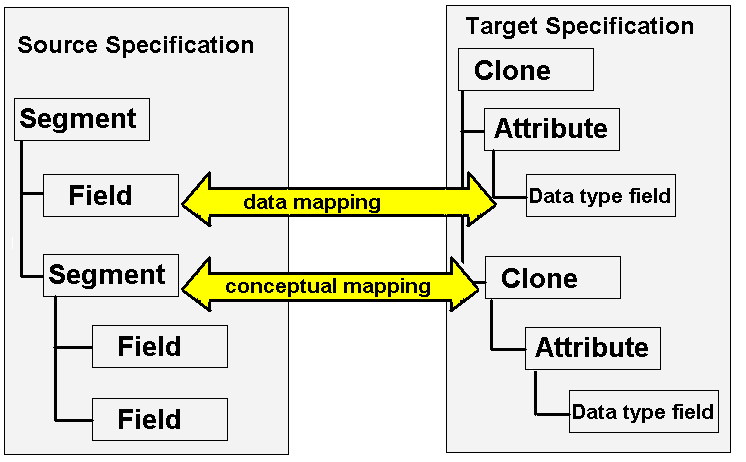
In general, during transformation, the caAdapter application will look at each of data type field defined in the target specification to see if any has been mapped to a field in the source specification. If so, it will examine whether the given data type field’s parent attribute or any of its parent attributes or clones above its parent attribute has any explicit conceptual mapping to the source field’s parent segment in the source specification. If so, this conceptual mapping will be utilized as factor 2; otherwise, an implicit mapping relationship is derived between the given data type field’s parent attribute and the given source field’s parent segment in the source specification. Similar implicit mapping relationships will be derived up to the source root segment and the target root clone, or until the first explicit conceptual mapping between a source segment and a target clone or attribute along the path back to the source root segment and the target root clone.

After either implicit or explicit conceptual mapping is decided, the cardinality information of the given data type field is analyzed to help determine the number of occurrences of message elements in messages generated. If the value is 1..1, according to its definition, one and only one message element will be generated and no further computation is needed. If the value is 1..\*, however, the cardinality from the implicit or explicit conceptual mapping will be referred to. If the cardinality value from the conceptual mapping is 1..1, it is obvious one and only one message element will be generated and will report warnings if more than one data record exist in the source data file. If the cardinality value from the conceptual mapping is 1..\*, the transformation service will create one HL7 v3 message for each occurrence of the data in the source data file.

Since the one-to-one cardinality relationship (1..1) creates one of the strongest bonds between the parent and child nodes, and given the priority in applying mapping rules (see the rule 1), a beginner of this tool may expect several validation error messages regarding mismatched “Multiplicity” or “Cardinality” issues. This is because he or she may have mapped, for instance, a source segment that has one-to-many cardinality, to a target clone or attribute that has only one-to-one cardinality, and the underlying source data of the source specification include more than one data instance of the given source segment.

Therefore, when creating a map file, it is important to first understand the inner structure of the HL7 v3 specification, especially the cardinality definition between parent and child clones and attributes. After that, it is best to start with the concept level relationship before mapping data relationships.

Figure Mappings for source to target specification



### Source Specification

This chapter assumes the source data format is a comma-separated values (CSV) file. A comprehensive explanation of the CSV file format could be found under the topic “CSV Data File” in the User Guide.

Briefly speaking, the CSV file format is a tabular data format that has fields separated by the comma character.

Example of a CSV file:

John, Doe, 5600 Fishers Lane, Rockville, MD, 20857

Jack, McGinnis, 456 Washington Blvd Suite 1000, Washington, DC, 20002

Each line is a segment containing a logical grouping of fields. Each segment may have one or more dependent child segments to handle one-to-many relationships between logical groups of data. In order to identify the segments for each logical record, segment identifiers are always the first element on each line.

Example of a CSV file:

PERS, John, Doe, 5600 Fishers Lane, Rockville, MD, 20857

PERSID, 2.16.840.1.113883.19.1, 12345

PERSID, 2.16.840.1.113883.19.1, 67890

PERS, Jack, McGinnis, 456 Washington Blvd Suite 1000, Washington, DC, 20002

PERSID, 2.16.840.1.113883.19.1, 24680

In the above example, PERS is the segment identifier showing the root of each logical record. The PERSID segment identifiers show how one logical record can have many child records. The mapping tool will interpret this CSV file as two logical records of PERS each with one to many PERSID segments.

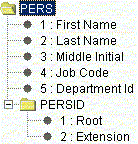


Figure Example of the parent-child relationship of source data

CSV source specification structures include a root segment with one or more child segments. A root segment and its child segments represent a single logical record. Each segment has one or more fields that represent elements of data. Segments may also contain one or more child segments that may also include child segments of their own. Mappings link both fields and segments to the XML elements in the target specification structure.

The *CSV Specification* tab allows users to reorder data fields, change structural relationships, and rename segments. Extra care should be taken when reordering segment structure and renaming segment names. The user should ensure that the source specification segment structure and names match the data file segment structure and names. The Validation action on source specification against CSV data function is very useful for uncovering errors of this type.

### Target Specification

This chapter assumes the target data format is an HL7 v3 XML message. An HL7 v3 specification is defined by the content of the HL7 .mif file for the target HL7 message type chosen by the user. A comprehensive explanation of the HL7 v3 specification and HL7 v3 Message can be found under the topic “HL7 v3 Specification” and “HL7 v3 Message” in the User Guide.

This quick start guide uses a hypothetical example HL7 v3 message for illustration. The example message has a single Act, Participation, Role and Entity.

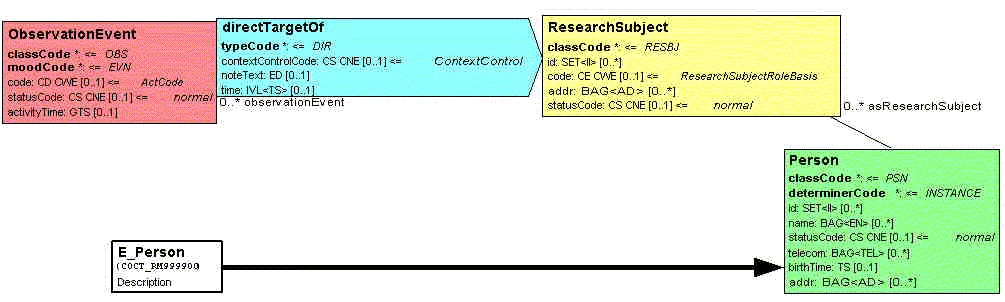


Figure COCT\_MT999900 Example Message

Target specification structures include the same elements as the message type. These elements include clones, attributes, and data type fields. Clones are classes derived from HL7 Reference Information Model (RIM) base classes. Each clone can have many child clones and many child attributes. An attributes is assigned a data type based on an independent specification of HL7 v3 data types.

Clones represent concept level hierarchy. Each clone may be associated to one or more child clones that may also be associated to child clones of their own. Some clones are associated to Common Message Element Types (CMETs). These CMETs are a set of one or more clones that represent reusable concepts and may appear in multiple places in a message. In the Mapping Tool users can create mapping links from source segments and fields to target clones, attributes, and data types.

“Person” is a clone

“id” is an attribute

“inlineText” is a data type

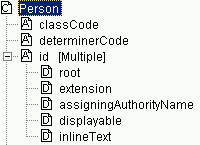


Figure The parts of a target specification structure

#### Defining Default Data

User-defined default values are constants assigned to data type fields. They are defined in the target specification, so they are pre-defined when it comes to the mapping. Please distinguish the “user-defined default value” from the “HL7 default value”, the latter of which is from the HL7 v3 message type defintion and is pre-defined and unchangeable. These user-defined default values allow users to assign values for attributes that may not be available from the source data. For example, if the ‘root’ for all user ids is common across the organization, this value can be entered in the target specification.

User-defined default values are assigned by using the *HL7 v3 Specification* tab. HL7 structural attributes and other elements that have their values fixed by the HL7 v3 standard cannot have default values defined.

User-defined default values are overridden by values from a mapped data source. While required attributes are often populated with default values specified in the HMD, optional ones are usually only populated when a map is present for that data type field or when a user-defined default value is specified. The table below shows the expected behavior for attributes that are mapped to a non-null CSV Value, mapped but to a CSV value that turns out the be null and unmapped data type fields.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Mapped | Mapped Null | UnMapped |
| Optional | CSV Value | CSV Value | Not Populated |
| Required | CSV Value | Default Value | Default Value |
| Mandatory | CSV Value | Default Value | Default Value |

Figure User Defined Default Value Behavior

“Mandatory” means that the value may not be NULL, unless its container (clone, attribute, etc.) is NULL. “Required” means values must be supported and may be NULL.

#### Units of Measure

Some HL7 v3 attributes contain units of measure properties. These units of measure must match those specified in the Unified Code for Units of Measure (UCUM). The Unified Code for Units of Measure is a code system intended to include all units of measure being contemporarily used in international science, engineering, and business. For a complete list, check the following resource:

The Unified Code for Units of Measure

http://aurora.regenstrief.org/UCUM/ucum.html

### Types of Mapped Relationships

#### Simple Mapping Relationship

Each link is a mapped relationship between source and target hierarchical trees. The combined rules for these mapped relationships define what the generated HL7 v3 message XML will look like.

Most mappings link data fields from a single segment to data type fields of an attribute. This document refers to these relationships as simple relationships because source cardinality and target cardinality have a one to one relationship.

In a simple relationship, one field in the source data file represents one data type field of an attribute on the target XML message.

#### Parent-Child Inverted Relationship

In a hierarchical tree structure, we define a node as a parent of another node if and only if the defined parent node hierarchically contains the other node. Similarly, we define the contained node is the child node of the aforementioned parent node. For example, in the Figure below, the “Organization” node of the source tree is the parent node of any “Doctor” node beneath it and any “Doctor” node is a child node of the “Organization” node.

Similarly, both source and target specification structures have a parent concept with its child concepts.

In some cases, a child in the source system will be mapped to a parent in the target system. And at the same time, the parent of aforementioned child in the source system will be mapped to a child of aforementioned parent in the target system.

For example, the source specification structure has an organization (parent) with one or more doctors (children). The target specification structure has doctors (parent) who are each associated with an organization (children). In this case each doctor has the same organization. This document refers to this type of relationship as parent-child inverted relationship.

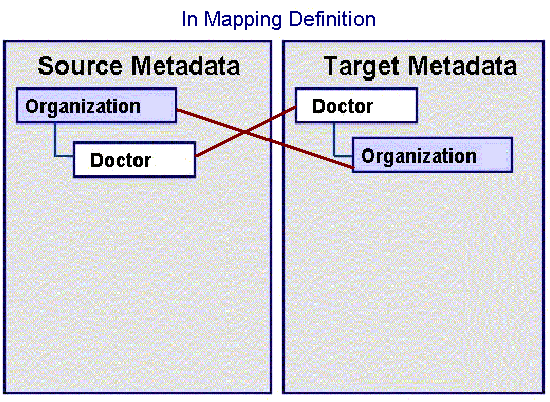


Figure . Example of a parent child inverted relationship in mapping definition

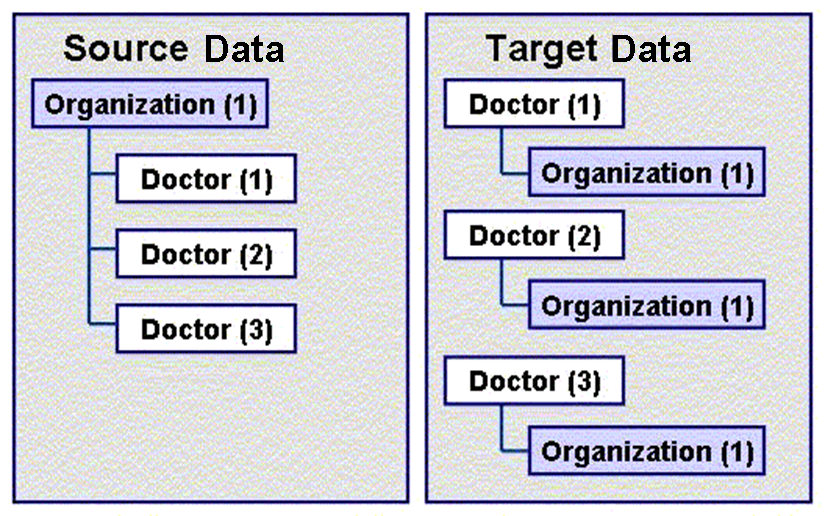


Figure . Example result of a parent child inverted relationship

In a parent-child inverted mapping, the rules state that:

**One parent source datum should be populated for each of the children**.

For example, a single organization that has three doctors in the source data will appear as three doctors each with the same organization in the target data. The parent element of source data, in this case the organization, is populated for each of the data type attributes in target.

#### Siblings

In , we defined the sibling relationship. With siblings, the mapping rule states that unless an explicit mapping is present for the common parent segment, one element of data in the source data file represents one data type attribute on the target file.

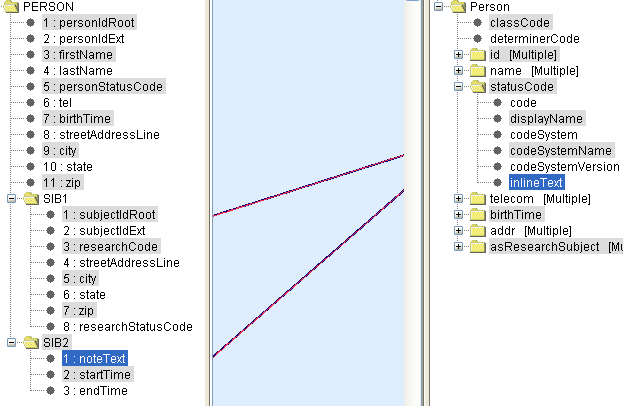


Figure Implicit Mapping on Sibling Structure

For example, in the figure above, PERSON.SIB1.subjectIdRoot from source is mapped to Person.statusCode.codeSystemName in the target and PERSON.SIB2.startTime is mapped to Person.statusCode.inlineText.

According to the sibling rule, Person.statusCode.codeSystemName and Person.statusCode.inlineText from the target are siblings under the Person.statusCode context.

As an extension to the Sibling definition, we also consider that both PERSON.SIB1.subjectIdRoot and PERSON.SIB2.startTime from the source share the same mutual root at PERSON.

To apply the mapping rule stated above, unless there exists an explicit conceptual mapping between PERSON from the source and Person.statusCode at target, 3 records will be generated, given 1 represents SIB1 data and the rest 2 represent information from each of SIB2 data, if the underlying CSV data has following value set.

|  |  |  |  |
| --- | --- | --- | --- |
| *Field Names* | … … | … … | … … |
| PERSON | … … | … … | … … |
| *Field Names* | **subjectIdRoot** | … … | … … |
| SIB1 | 20030102 | … … | … … |
| *Field Names* | … … | **startTime** | … … |
| SIB2 | … … | B | … … |
| SIB2 | … … | J | … … |

The generated result contains with 3 records. Note that the first record does not have value for “inlineText”, while the second and third records do not have value for “codeSystemName”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Field Names* | … … | codeSystemName | … … | inlineText | … … |
| Record 1 | … … | 20030102 | … … |  | … … |
| Record 2 | … … |  | … … | B | … … |
| Record 3 | … … |  | … … | J | … … |

On the other hand, if there does exist an explicit mapping between PERSON from source and the Person.statusCode from the target, the cardinality definition of the common parent, i.e., the PERSON element from the source will have weight on the number of records being generated. For instance, if the cardinality of the PERSON element from the source is 1..1, one and only one record will be rendered in the result, no matter how many source data record of each SIB1 or SIB2 exist, although a validation error will be reported on the mismatched cardinality because more than one data from SIB2 exist.

If the cardinality of the PERSON element from the source is 1..\*, however, the actual number of data on SIB1 and SIB2 in the real data file will rule.

#### Conceptual Relationship

All mappings have an explicit or implicit conceptual relationship between source and target elements. An implicit relationship is defined by the nature of the link. For example, a child segment mapped to child clone implies the existence of an implicit relationship between their common parents. The mapping tool also supports explicit mapping, where the user creates a link between a source segment and a target clone, or between a source segment and a target attribute. Under the explicit conceptual mapping, unless the target has one-to-one (1..1) cardinality relationship with its parent, the cardinality from the source will help govern the number of records of the targeted node being generated, given its underlying children nodes’ mapping structure.

### Transformation Service

The caAdapter Mapping Tool uses the transformation service to determine target message structure based on mapping rules. These rules are based on the types of mapped relationship used in the .map file.

The Transformation Service builds the HL7 v3 XML message. It creates the message by traversing an object graph of HL7 clone objects. It then creates and populates these clone data objects based on the mapping rules.

#### Mapping Rules

In order for clones to be aggregated, a relationship needs to be established between the target clone and a source segment. The following rules define how these relationships are derived:

* The transformation service checks for conceptual mappings to the target clone in question. These are treated as having an explicit relationship between the target clone and the source segment.
* The transformation service checks for data mappings to the target clone in question. All source segments that are mapped to attributes of this target clone are treated as having an implicit relationship between the target clone and those source segment(s). Then the common parent of these segments is derived.
* The number of target clones created equals the number of corresponding implicit or explicit source segments.
* Child target clones will inherit the implicit or explicit source segments relationships from their parent clones. In these cases a single clone is created.

#### Mapping Functions

Functions are inline processes that are used to provide certain transformations to the data during map processing. For example, the concatenate function can combine two fields in the source data file into a single data type attribute.

Only fields from a single segment can be used as input to a function.

Functions can be chained together to offer a series of actions to one target attribute. In these cases all inputs to the chain of functions have to be from the same source segment.

## Mapping Example

This section will provide an example and walk through the steps required to transform a segment-based CSV data file to HL7 v3 mappings. Files for this example are located in the “Scenario8” directory under the “Mapping Scenarios” folder.

This example will

* Generate the segment-based CSV data from the sample output data;
* Generate CSV segment specification, a.k.a. SCS file, from the segment-based CSV data;
* Generate HL7 v3 specification, a.k.a. H3S file after some customizations of clone structure and filling in user default values, if any, from the pre-defined COCT\_RM999900 HMD template;
* Create mapping specification, a.k.a. Map file, to design the mapping relationship between the SCS and H3S files, via the caAdapter mapping panel;
* Render the HL7 v3 messages given both the sample segment-based CSV output data and the mapping specification as inputs;

The sample output data is organized by patient record (PERSON). Each patient record (PERSON) participates as a research subject (RESEARCHSUBJECT) as the target (DIRECTTARGET) of an observation (OBSERVATIONEVENT).

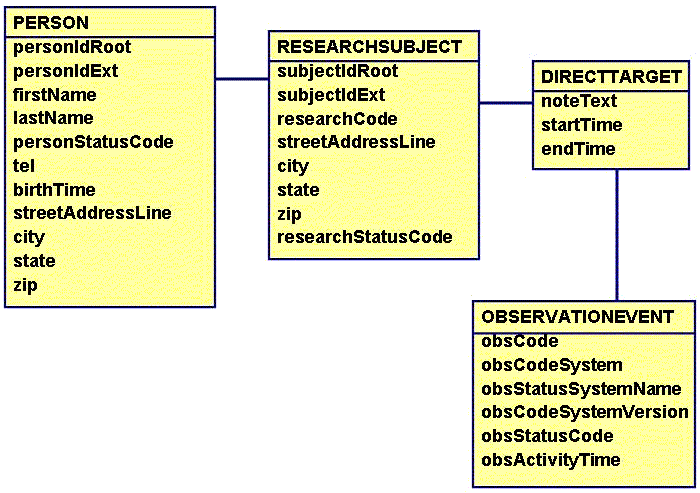


Figure Sample Output Data

Using the sample output data, a CSV data file is generated. Each one-to-many relationship becomes a segment with the PERSON segment being the root.

PERSON, 2.16.840.1.1138..., 12345, John, Smith, 01, 800-555-9999, 20030102, 456 Washington Blvd, Washington, DC, 20002

RESEARCHSUBJECT, 2.16.840.1.113883.19.1, 12345, ABC123, 456 Washington Blvd, Washington, DC, 20002, 01

DIRECTTARGET, This is target 1, 20030102, 20030704

OBSERVATIONEVENT, ABC123, 2.16.840.1.113883.5.4, caDSR, 3.01, 01, 20030607

DIRECTTARGET, This is target 2, 20030704, 20040214

OBSERVATIONEVENT, LMO456, 2.16.840.1.113883.5.4, caDSR, 3.01, 01, 20030823

DIRECTTARGET, This is target 3, 20050315, 20050926

OBSERVATIONEVENT, XYZ789, 2.16.840.1.113883.5.4, caDSR, 3.01, 01, 2005076

Sample Output CSV File

### Source Meta Specification

The CSV segment specification, a.k.a. SCS file, is created from the sample output CSV file.

Field names and segment structure are defined by use of the “scs” panel in the caAdapter application. Users can rename names of field and segment elements. Users can also modify parent-child relationship between segment and field or between segments by dragging and dropping source segments or fields onto the targeted segment location. In addition, users may reshuffle the order of field elements under the same segment, to furnish alternative layout of data structure. These structural relationships (i.e. parent-child, siblings, etc.) are essential, because combined with actual CSV data, they will contribute to the way how the transformation service interprets the mapping relationships in generating the target HL7 v3 messages.

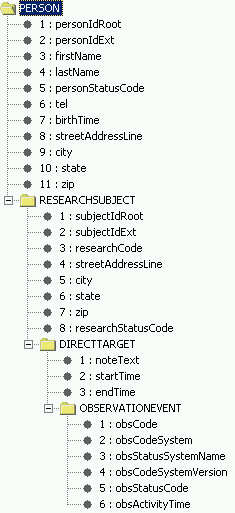


Figure Example of CVS Source Specification

In our example each of the source segments is a child of the segment that came before it.

### Target Meta Specification

The example target specification specification, a.k.a. H3S file, includes default values defined for some fields or default clone layout per generic message type. For example, observationEvent code has a default codeSystem already applied. The codeSystem is the same for all messages created with this h3s specification file. Multiple instances of clones or data types could have also added to the target specification.

### Mappings

Each transformation from the source specification (SCS file) to the target HL7 v3 specification (H3S file) in this example requires an explicit mapping relationship. In the example, an explicit map exists between DIRECTTARGET and the HL7 v3 clone of asResearchSubject.

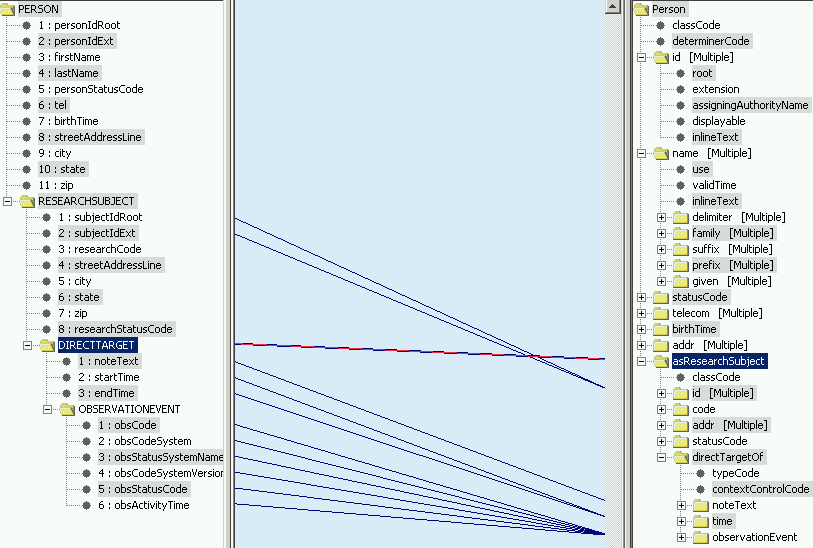


Figure Example Mapping

The rest of the mapping is simple one to one maps, between the source specification and target specification.

### The Generation of the HL7 v3 messages

After the map file is created, users could utilize the HL7 v3 message generation utility in the caAdpater application to render the HL7 v3 message instances for the mapped HL7 v3 specification. Following is the example HL7 v3 message.

<?xml version="1.0" encoding="UTF-8"?>

<COCT\_MT999900.Person classCode="PSN" determinerCode="INSTANCE">

<asResearchSubject classCode="RESBJ">

<id root="2.16.840.1.113883.19.1" extension="12345"/>

<code/>

<statusCode/>

<directTargetOf typeCode="DIR">

<noteText>This is target 1</noteText>

<time>

<low value="20030102"/>

<high value="20030704"/>

</time>

<observationEvent classCode="OBS" moodCode="EVN">

<code code="ABC123" codeSystem="2.16.840.1.113883.5.4" codeSystemName="caDSR" codeSystemVersion="3.01"/>

<statusCode code="01" codeSystem="2.16.840.1.113883.19.1"/>

<activityTime value="20030607"/>

</observationEvent>

</directTargetOf>

</asResearchSubject>

<asResearchSubject classCode="RESBJ">

<id root="2.16.840.1.113883.19.1" extension="12345"/>

<code/>

<statusCode/>

<directTargetOf typeCode="DIR">

<noteText>This is target 2</noteText>

<time>

<low value="20030704"/>

<high value="20040214"/>

</time>

<observationEvent classCode="OBS" moodCode="EVN">

<code code="LMO456" codeSystem="2.16.840.1.113883.5.4" codeSystemName="caDSR" codeSystemVersion="3.01"/>

<statusCode code="01" codeSystem="2.16.840.1.113883.19.1"/>

<activityTime value="20030823"/>

</observationEvent>

</directTargetOf>

</asResearchSubject>

<asResearchSubject classCode="RESBJ">

<id root="2.16.840.1.113883.19.1" extension="12345"/>

<code/>

<statusCode/>

<directTargetOf typeCode="DIR">

<noteText>This is target 3</noteText>

<time>

<low value="20050315"/>

<high value="20050926"/>

</time>

<observationEvent classCode="OBS" moodCode="EVN">

<code code="XYZ789" codeSystem="2.16.840.1.113883.5.4" codeSystemName="caDSR" codeSystemVersion="3.01"/>

<statusCode code="01" codeSystem="2.16.840.1.113883.19.1"/>

<activityTime value="2005083"/>

</observationEvent>

</directTargetOf>

</asResearchSubject>

</COCT\_MT999900.Person>

## Conclusion

The example shows the steps to generate HL7 v3 messages from a segment-based CSV data file. To build the mapping file, users need to have both the SCS and H3S files available. The SCS and H3S files could be created and modified by utilizing the SCS and H3S modules, respectively, in the caAdapter application. Once the map files have been created, HL7 v3 message instances will be generated via HL7 v3 message module for the mapped HL7 v3 specification.

## A Word about HL7 v2.x to HL7 v3 Mapping

caAdapter 4.0 contains a module for generating an SCS from a set of specifications for HL7 v2.x elements (messages, segments, data types and vocabularies). This modules also has the ability to convert an HL7 v2.x message instance into a CSV file that complies with the v2.x-based SCS. Thus the mapping of v2.x elements to HL7 v3 elements is accomplished using the CSV to HL7 v3 Mapping module. All the same business rules apply in this context as for a normal CSV to HL7 v3 mapping that is not based on a v2.x-based SCS. Depending on the complexity of the message, it is quite likely that analysts will need to be very familiar with the conceptual mapping described above to successfully create a v2.x-based CSV to HL7 v3 mapping. In such a case, trial and error is the best teacher. It is recommended that analysts test the transformation frequently as they build the mappings so that they learn how conceptual mapping guides the transformation engine.

# CSV to RDS (SDTM) Mapping

## Introduction

This module allows mapping existing data structures to generate Regulatory-type Data Submission (RDS) files. This version of caAdapter only supports creating SDTM files from a database or a CSV source. The user can use the Mapping Tool to map database tables, or a CSV files, structure to one or more SDTM domain structure. The transformation feature then transforms and generates corresponding SDTM domain files.

For consistency, here we list a group of generally used terms that may be referenced in this context.

|  |  |
| --- | --- |
| Term | Explanation |
| CSV Specification | A comma-separated value (CSV) specification that presents the segment-based structure of a series of CSV data files |
| Source Specification | In current context, it refers to the CSV Specification |
| define.xml | The XML-based specification that is produced by CDISC to describe the structure of an SDTM submission. This may also include local customizations made by a sponsor to tailor the submission within allowable parameters to their specific needs. |
| SDTM Specification | In current context, it refers to the define.xml |
| Target Specification | In current context, it refers to the define.xml |
| Map Specification or Map File | An XML-based specification that records the mapping relationship between a source and target specification |
| CSV data file | A segment-based CSV data file that may conform to a certain CSV Specification hierarchically |
| Source data file | In current context, it refers to CSV data file |
| SDTM text file | The outcome of the transformations service, i.e. the data file that conforms to the target specification, the define.xml |
| Target data file | In current context, it refers to SDTM text file |

## Mapping Rules

The mapping capability of the CSV to SDTM module is constrained for this release and serves as a very simple prototype. Thus the mapping rules are simple:

* SCS fields may be mapped to SDTM domain variables;
* Neither SCS segments nor SDTM domains may be mapped;
* A given SCS field may be mapped to more than one domain variable;
* Only the Demographics (DM) domain may be mapped for this release (other domains are displayed for informational purposes only, and will be not be “mappable” until a later release);
* Mapping fields from a single segment to the same domain will result in the same number of SDTM text file records as in the input CSV file;
* Mapping fields from parent and child segments to the same domain will result in one DM record in the generated SDTM text file for each child (e.g. if a CSV file has a parent segment that has 5 instances of the child segment following it, then there will be 5 DM records in the SDTM text file generated using this CSV and mapping file pair).

Note that currently there are no validation features for this mapping and transformation, so it is up to the analyst to ensure maps and resulting SDTM text files are valid. This also includes format and vocabulary validation as well.

## Mapping Example

The following example shows how a CSV to SDTM mapping may look. This example shows the fields of parent and child segments mapped to the variables of the DM domain.

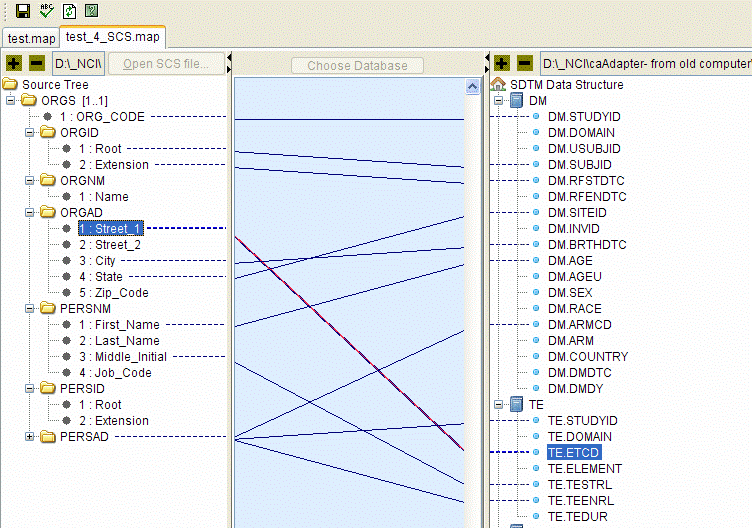


Figure Example of CSV to SDTM Mapping

In this case, the data that would be common to all records is part of the parent segment, ORGS, and the data that is record-specific is part of the child segment, PRSNM, CSV fields are mapped to the DM variables and the transformation engine will handle the fact that there is a one-to-many relationship between the parent and child segments and create one DM record for per child segment, reusing the data from the parent segment in all DM records.

## Recommendations for CSV Specification

Given that some SDTM data elements will be the same for all or a group of records, it is recommended that common data elements be grouped into a single segment that then can be created once while data that differs from record to record is placed on a child segment that follows its parent. Alternatively, for simplicity sake, a completely flat file structure could be created that has all data elements generated on a single segment.

# Model Mapping Service

## Introduction

The caAdapter application contains one module used in Object to Database mapping. This module uses an XMI export of an Enterprise Architect project. The export may contain either just an object model or both an object model and corresponding data model. caAdapter also supports an object model created by ArgoUML. If the export only contains the object model, the Object to Database Mapping module allows the user to import the data model from the database directly. This module then provides the ability to map the objects to the tables. The mapping service does not have a corresponding transformation module as do the other mapping services.

The caAdapter Model Mapping Service provides the following functionalities:

* Parse and load data model and object model from an xmi file, or uml file
* Drag and drop mapping between an object model and a data model
* Add SDK required tags and tag values into an xmi file
* Generate Hibernate mapping file

## Basic Mapping

caAdapter supports three basic mapping types. They are: (1) dependency mapping, (2) attribute mapping, and (3) association mapping).

### Dependency Mapping (Object to Table)

A dependency mapping is a mapping between an object and a table. Perform the following steps to create a dependency mapping.

1. Select a source element from the Object Model on the right. The example in Figure 16 shows HealthcareSite. Click and drag to HealthcareSite to HEALTHCARE\_SITE in the Data Model.
2. A mapping line between HealthcareSite in the Object Model and HEALTHCARE\_SITE in the Data Model should now be visible. Dependency mapping lines are color-coded green.

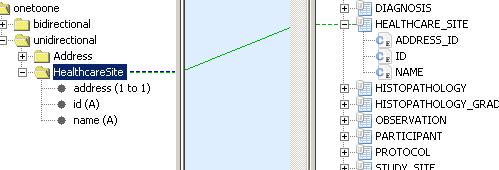


Figure 16Dependency Mapping

#### Attribute Mapping

An attribute mapping is a mapping between an attribute in the object model and a column in the data model. (Before any attribute mapping can be performed, users have to complete dependency mapping first) Perform the following steps to create an attribute mapping.

1. The example in Figure 17 shows the attribute id (A) for the class HealthcareSite. Select 'id (A)' in the Object Model and drag it to ID in the Data Model.
2. A mapping line should be visible between the attribute and column. Attribute mapping lines are color-coded blue. Repeat this for 'name (A)' to NAME.

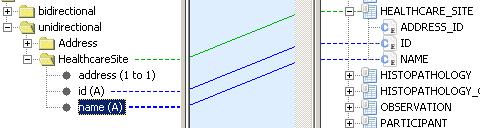


Figure 17 Attribute Mapping

1. If the object has not already been mapped to the table, an attempt to map the object’s attributes to the table’s columns will result in an error message (Figure 18)



Figure 18 Attribute Mapping error message

### Association Mapping

An association mapping is a mapping 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 create an association mapping.

1. First create a dependency mapping between the object model and the data model. For example, in Figure 4-4 the green line shows a dependency between 'HealthcareSite' and ‘HEALTHCARE\_SITE’.
2. Map 'id (A)' to ID and 'name (A)' to NAME.
3. Click and drag 'address (1 to 1)' to ADDRESS\_ID. When complete, the final result should look like Figure 19. Association mapping lines are color-coded red.

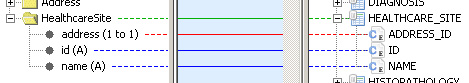


Figure 19 Association Mapping

For additional information on the different types of association mapping, please reference caAdapter 4.1 User’s Guide