caAdapter

Design Document

Version 1.3

August 3, 2006

National Cancer Institute
6116 Executive Blvd.
Rockville, MD 20852

1. Document Overview

This chapter contains the following topics:

1.1 Change Record

Date	Author(s)	Document Version	Change Reference (Major Changes)
8/17/05	Eric Chen, Matt Giordano, Wendy Ver Hoef, Jayfus Doswell, Scott Jiang	1.2	Initial Document
8/18/05	Dan Grandquist	1.2	Add Reports and Proprty Design
02/05/2006	Matt Giordano, Scott Jiang	1.2	Update prior to 1.2 release
3/23/2006	Eric Chen, Scott Jiang	1.3	Add HL7 v3 API Upgrading

1.2 Reviewers

Name	Position	Document Version	Date Reviewed
Sichen Liu	Scientist, Clinical Informatics, NCICB	1.2	TBD
Sharon Settnek	Program Manager, SAIC	1.2	TBD

Table 1-1 Reviewers

2. Introduction

Topics in this chapter include:

2.1 Purpose of Document

The purpose of this design document is to provide the design of the caAdapter application of the National Cancer Institute's Center for Bioinformatics (NCICB). This document captures the important design decisions of several distinct but interrelated software artifacts including: architectural design, subsystem designs, object and graphical user interface (GUI) designs, and file format designs. It also provides the basis for the software implementation which is the next stage in the software development life cycle.

2.2 Overview

The caAdapter application will be developed as an open source Graphical User Interface (GUI) application that enables analysts and database engineers, who are knowledgeable about HL7, to create mappings from a variety of data formats to an HL7 version 3 (v3) message structure (for example, Extensible Markup Language (XML)). The foreseeable formats that will be accepted by the mapping tool include, but are not limited to Comma Separated Value (CSV), non-HL7 XML, database data, HL7 v2 messages, and Java object formats.

The first phase of the mapping tool development shall focus on CSV to HL7 v3 mapping. In the first phase, this mapping tool will be designed as an easy-to-use GUI for mapping CSV clinical data to an equivalent target HL7 v3 format. Corresponding changes will also be made to the caAdapter runtime engine to facilitate the conversion of actual CSV data to HL7 v3 XML instances. Later phases of the project will extend the mapping tool and runtime engine capabilities to handle other data formats (for example, tab delimited) for mapping to an HL7 v3 message.

2.3 Scope

In this phase, the mapping framework, CSV file processing capability, Reference Information Model (RIM) graph generator, error and log handling, and the GUI functionality will be implemented. This

document does not cover implementation details and it is not intended to be used as a User's Guide. Furthermore, this design document is subject to updates throughout the subsequent development phases.

2.4 Design Principles

Following are design principles that shall be followed during design updates and implementation phases.

- **Minimal v3 Specification Interpretation**
  The software classes shall reflect HL7 v3 specifications as directly as possible.

- **Extensible Message Format**
  The collection of software classes shall be designed to incorporate a **generic mapping solution** such that any message type including, but not limited to CSV, non-HL7 XML, database data, HL7 V2 messages, and Java object formats can be mapped to an HL7 v3 message.

- **Intelligent Use of HL7 v3 Specifications**
  The resulting software shall utilize the HL7 v3 specifications and resources (for example, Hierarchical Message Descriptions (HMDs)) as intended.

- **Scalable Design**
  The resulting software and hence the collection of classes shall scale to an increasing number of message nodes while also maintaining optimal performance.

- **Interoperable Design**
  Resulting software shall be designed in an abstract and **de-coupled** manner such that the data model (that is, mapping model) may interface with multiple views with minimal modification as improvements are made to the human computer interface.

- **Error Handling**
  Software classes shall utilize effective error handling strategies that identify and expose descriptions about problems within a class's structure. Additionally, these classes shall broadcast to the development team the problems identified so that it may be addressed based on its severity. During implementation, disciplined unit testing will be performed by developers to identify early bugs. Developers will then fix bugs before each software release to minimize bugs during production. Additionally, **general Exception** classes shall not be used. Only specific Exceptions that catch an identified error tested during unit testing of the software shall be used.

- **System Performance**
  The resulting software system shall perform optimally. Specifically, each subsystem and its respective components shall operate in an efficient manner and satisfy user expectation as a reflection of well thought out software design. Additionally, the software shall seriously address **system performance** in software elements and design processes ranging from class structure and class communication to variables chosen during implementation. As a result, the

expected outcome is that the software system will perform optimally and to the expectation of the end-user.

2.5 Audience

This document is intended for consumption primarily by:

- Software engineers who will be implementing the software
- System administrators
- Quality Assurance (QA) engineers and testers

2.6 Related Documents and References

Listed below are documents that are related to the caAdapter Project:

1. *caAdapter Use Cases*
2. *caAdapter v1.3 Requirement*

2.7 Important Websites

- HL7
  http://www.hl7.org/

- HL7 Java Special Interest Group (JavaSIG)
  http://www.hl7.org/Special/committees/java/index.cfm

- HL7 Common Terminology Service (CTS)
  http://informatics.mayo.edu/index.php?page=11

3. Architecture Design

The purpose of this chapter is to communicate the caAdapter architecture design, its corresponding subsystems, and related components. Topics in this chapter include:

3.1 Software Architecture

Computer software architecture defines the structure of a software system, which includes software components, the externally visible properties of those components, and the relationships among them [L. Bass, P. Clements & R. Kazman, Software Architecture in Practice, Addison Wesley 1998]. Software architectures can provide a basis for the capture and subsequent reuse of design knowledge. Consequently, the goal of software architecture is to allow the design of a system to take place at a higher level of abstraction capturing the most important top-level decisions about the overall structure and associated behavior of one or more software components. This section provides a high-level overview of how the functionality and responsibilities of the system were partitioned and then assigned to subsystems or components. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together to provide the desired functionality. Specifically, this document describes:

- The major responsibilities that the software must undertake (such as processing or messaging) and the various roles that the system (or portions of the system) must play.
- How the system is decomposed into its components/subsystems (identifying each top-level component/subsystem and the roles/responsibilities assigned to it).
- Information on the access/persistence mechanisms.
- How the higher-level components collaborate with each other in order to achieve the required results.

3.2 Architecture Diagram

An architecture diagram is used to illustrate an architecture design decision and the structure of inter-relating software components.

Figure 3-1 illustrates the caAdapter core engine architecture design including its HL7 v3 API and validation component.

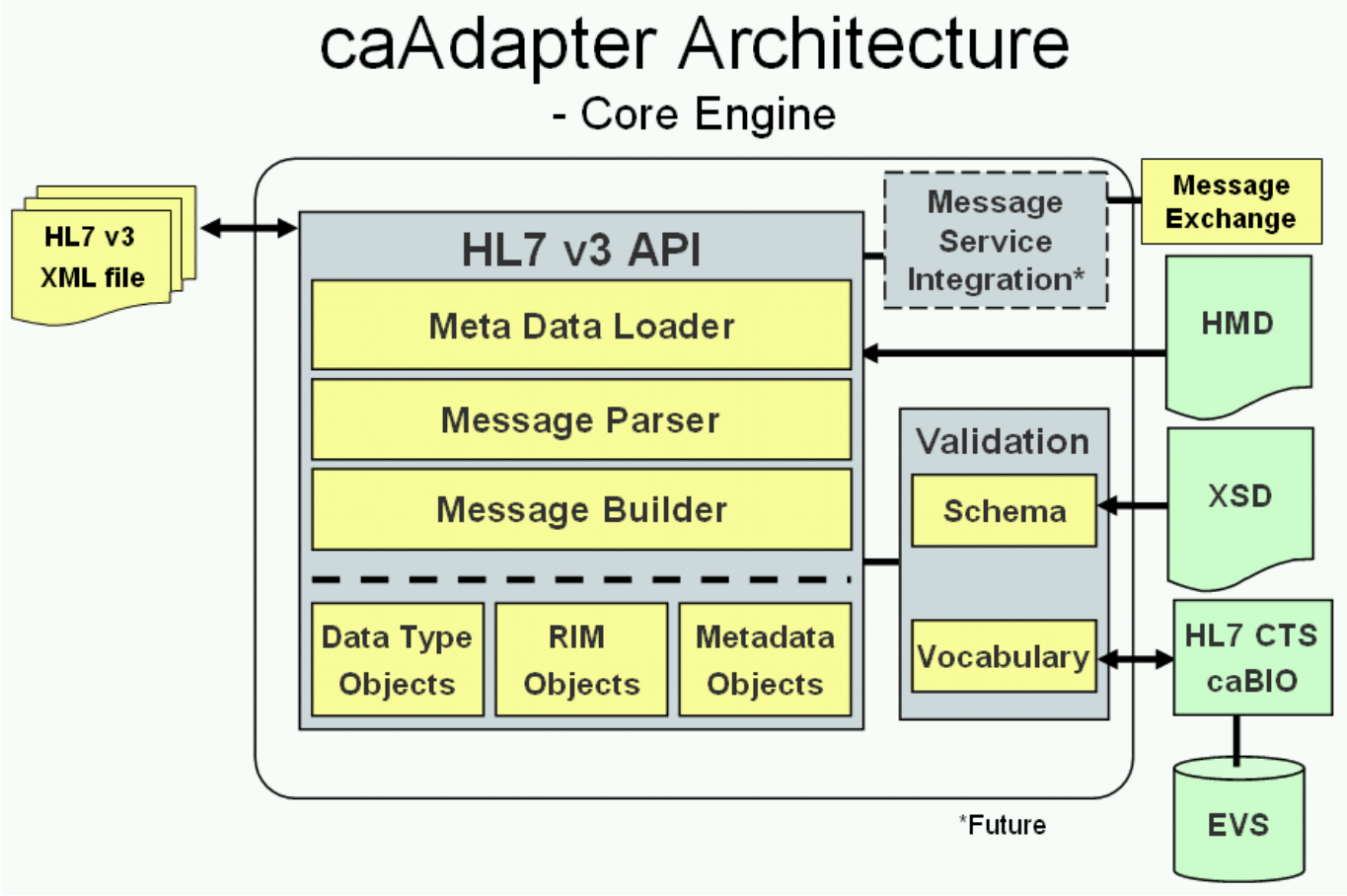


Figure 3-1 caAdapter High Level Architecture-Core Engine

Figure 3-2 illustrates the caAdapter mapping tool architecture. For more information, see section *3.3.3 Mapping Subsystem* on page 11.

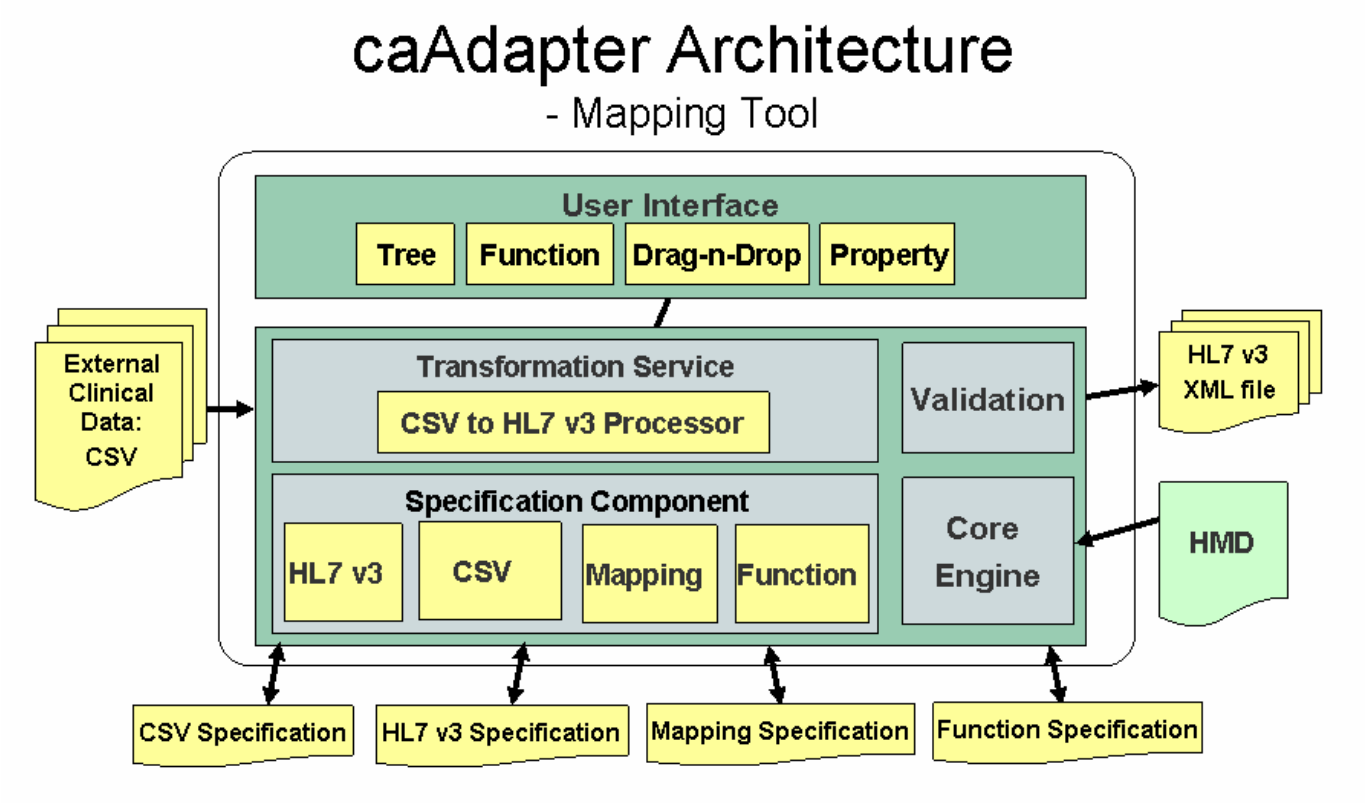


Figure 3-2 caAdapter High Level Architecture – Mapping Tool

3.3 Subsystem Overview

The goal of the caAdapter 1.2 software system is to provide an open source solution that facilitates the creation of HL7 v3 XML messages through a generic mapping solution. This section discusses each subsystem design (as illustrated in *Figure 3-1* and *Figure 3-2* on page 10) that is implemented in the caAdapter.

3.3.1 GUI Subsystem

The GUI Subsystem presents a graphical interface that allows users to define, modify, export and import source (csv) and target (hl7) specifications. The GUI enables end-users the ability to drag and drop between source and target file formats to realize a mapping file structure, and consequently, produce an HL7 message. For more information, see section *4 GUI Subsystem* on page 13.

3.3.2 CSV Subsystem

There are several objectives for the CSV Subsystem:

1. to read and write CSV specifications (build and parse).
2. to parse and validate CSV data based on a CSV specification

For more information, see section *5 CSV Subsystem* on page 29.

3.3.3 Mapping Subsystem

The Mapping Subsystem contains three lower level subsystems:

1. Mapping Specification - to read and write map files (build and parse)
2. Map Processor - to process mapfiles into Clone data objects
3. RIM Graph Generator - to generate RIM objects based on Clone data objects

For more information, see section *6 Mapping* on page 32.

3.3.4 Function Subsystem

The Function Subsystem provides the ability to register functions (for example, concatenation) and execute functions when generating a HL7 messages.
For more information, see section *7 Function Subsystem* on page 34.

3.3.5 HL7 v3 Subsystem

The HL7 v3 Subsystem facilitates the specialization of HL7 metadata that reflects an end-user's customization of the specification.
For more information, see section *8 HL7 v3* Subsystem on page 35.

3.3.6 HL7 v3 API

The HL7 v3 API is HL7 v3 messaging framework is based on an object-oriented data model which including Reference Information Model, data types, and HL7 v3 Meta objects. It can parses HL7 v3 to RIM object graph as well as build HL7 v3 messages from the RIM object graph.
For more information, see section *8 HL7 v3* Subsystem on page 35.

3.4 Transformation Service Sequence Diagram

Figure 3-3 illustrates a Unified Modeling Language (UML) sequence diagram depicting the interaction between objects and messages designed for building an HL7 v3 message.

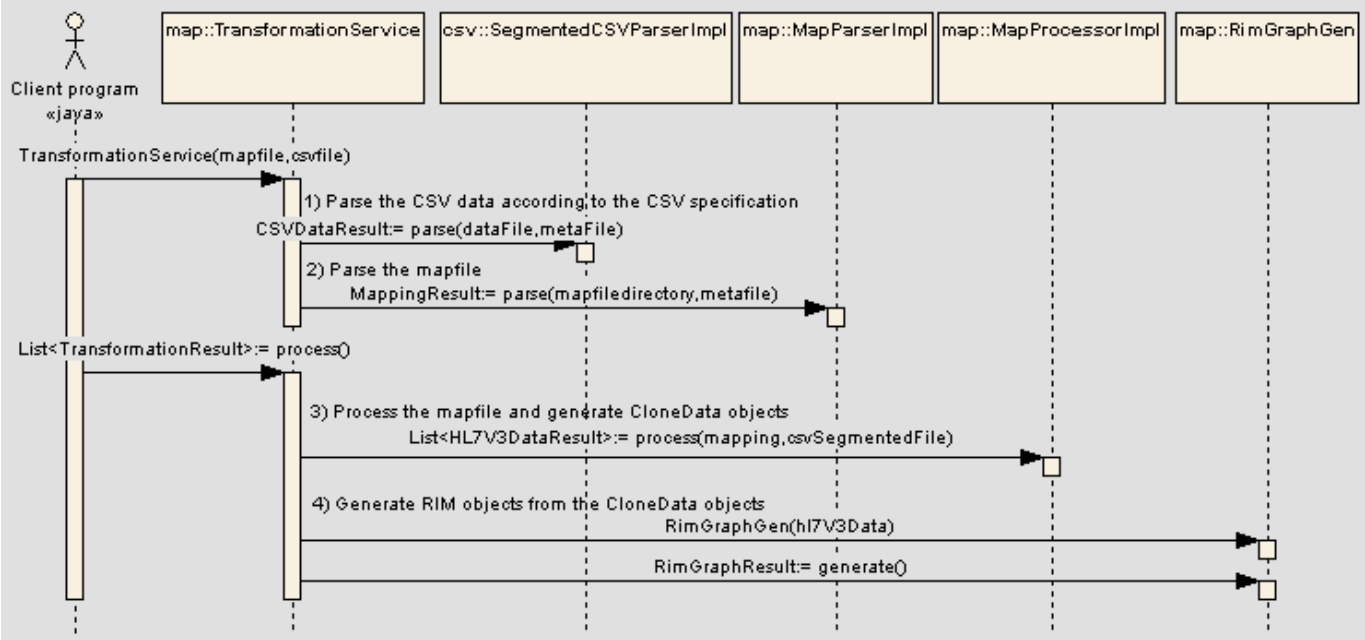


Figure 3-3 Sequence Diagram for TransformationService

In *Figure 3-3*, the `TransformationService` object acts as a coordinator object, making a series of message calls to other objects in the following order.

1. A `TransformationService` object is instanciated with a csv datafile and a map file.
2. The csv file parsed according to the csv specification that is located within the mapfile.
3. The map file is then parsed into the Mapping object graph.
4. The in memory csv and mapping information is passed to the MapProcessor which traverses the data and generates CloneData objects based on the mapping rules.
5. The clone data objects is passed to the RimGraphGen(erator) which generates RIM objects which will then be sent to the HL7 XML message builder. (the XML builder is not present in the above sequence diagram)

4. GUI Subsystem

The GUI subsystem is made up of a human computer interface (see section *9 HL7 v3 API*
The purpose of this chapter is to present various aspects of knowledge related to HL7 v3 API that caAdapter tool depends upon.

-
- Topics in this chapter include:
- Reference Information Model on this page
- Document Overview on page 41

Meta Data Model on page 46

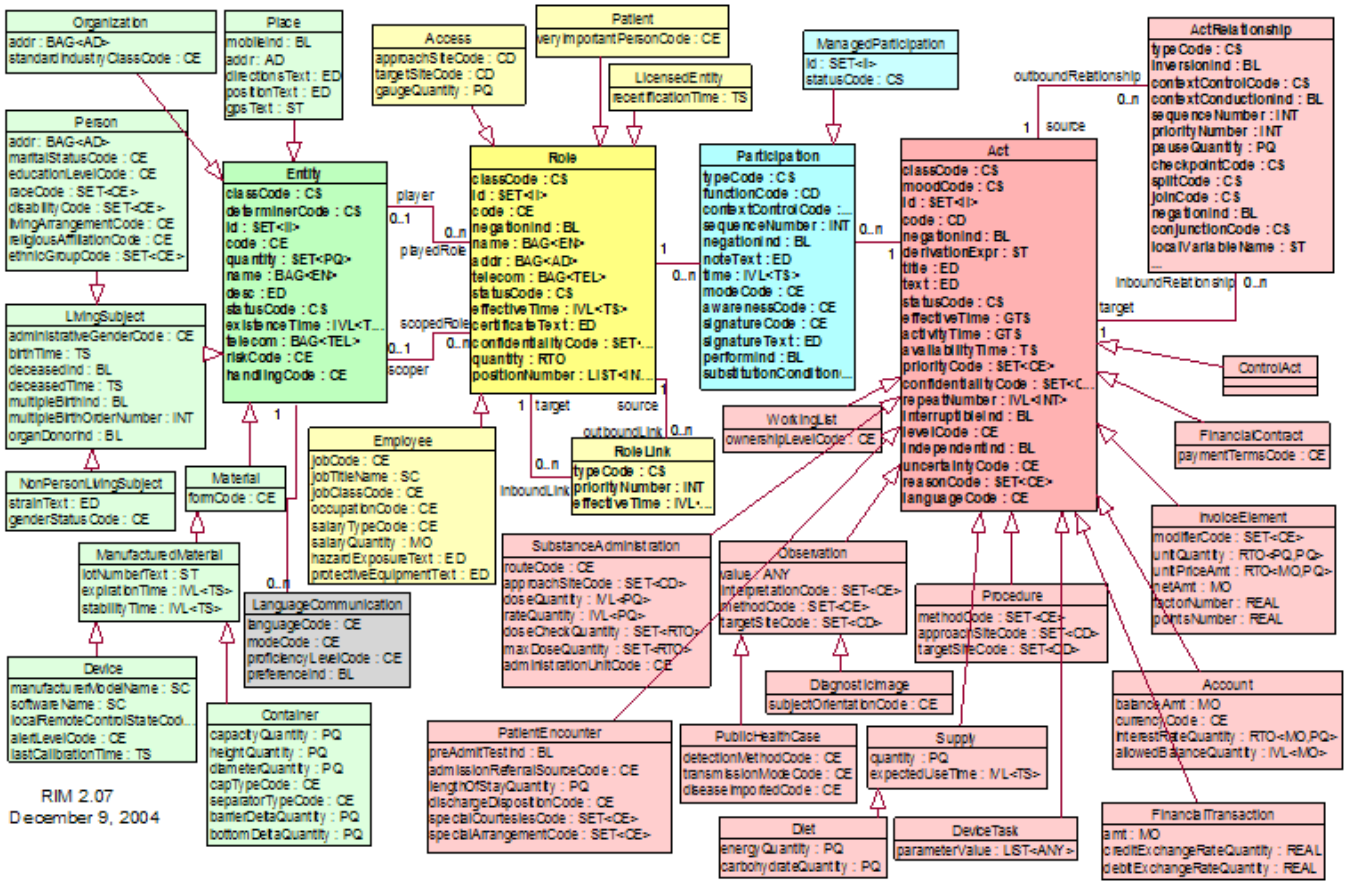
4.1 Message Parser on page 46

Message Builder on page 47

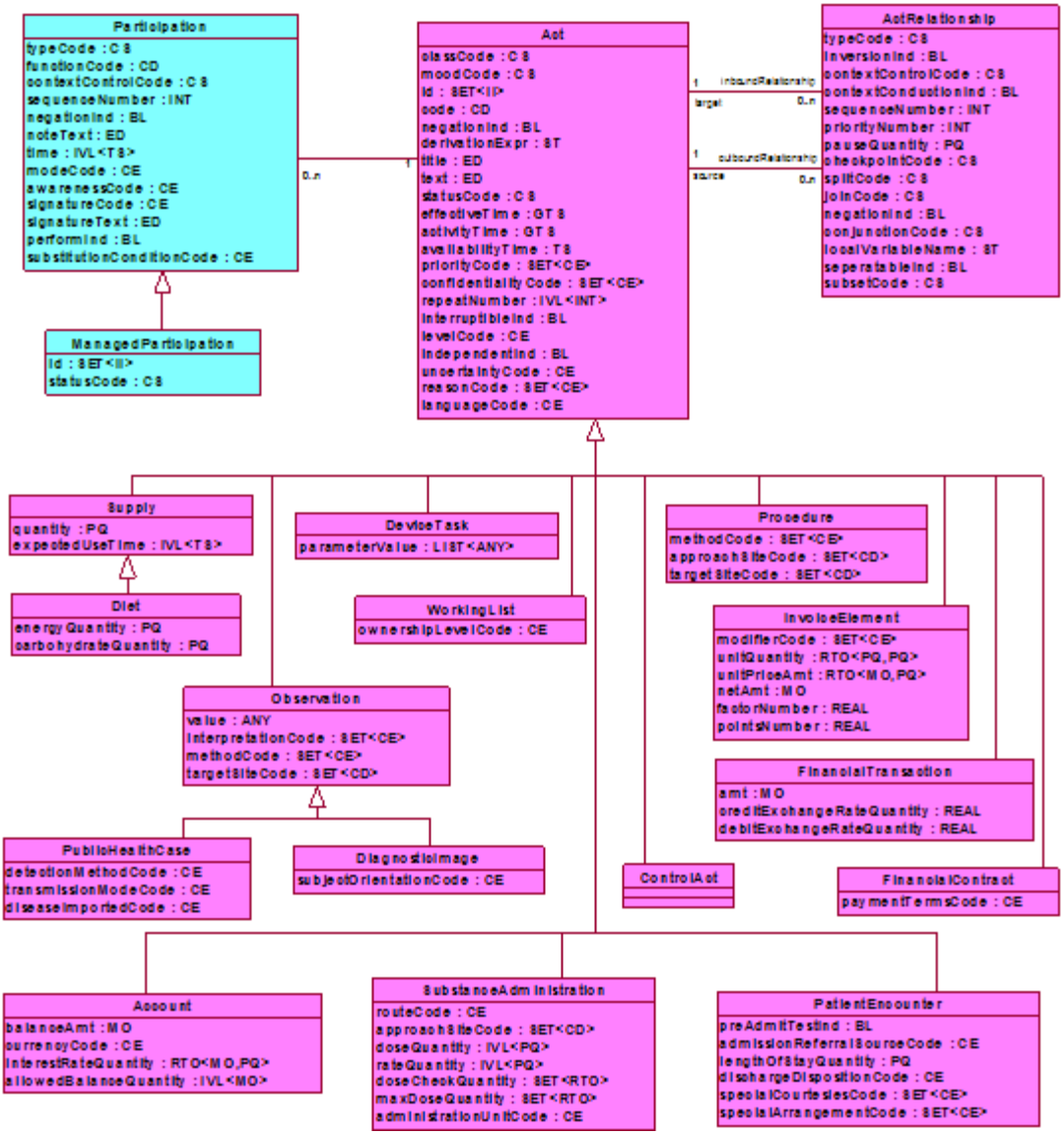
Reference Information Model
The source files for these classes are automatically generated from the XSLT transform of definition files. The RIM classes are based on high level abstractions such as patients, observations, procedures, roles, acts, etc.

At current release, the team is responsible to migrate the current support of RIM 2.02 to the latest Normative release as of RIM 2.07. A comparative research has been conducted to collect the gap analyis on accumulative variations since RIM 2.02 till RIM 2.07. A separate spreadsheet document has been constructed to denote corresponding changes in several areas that are covered below.

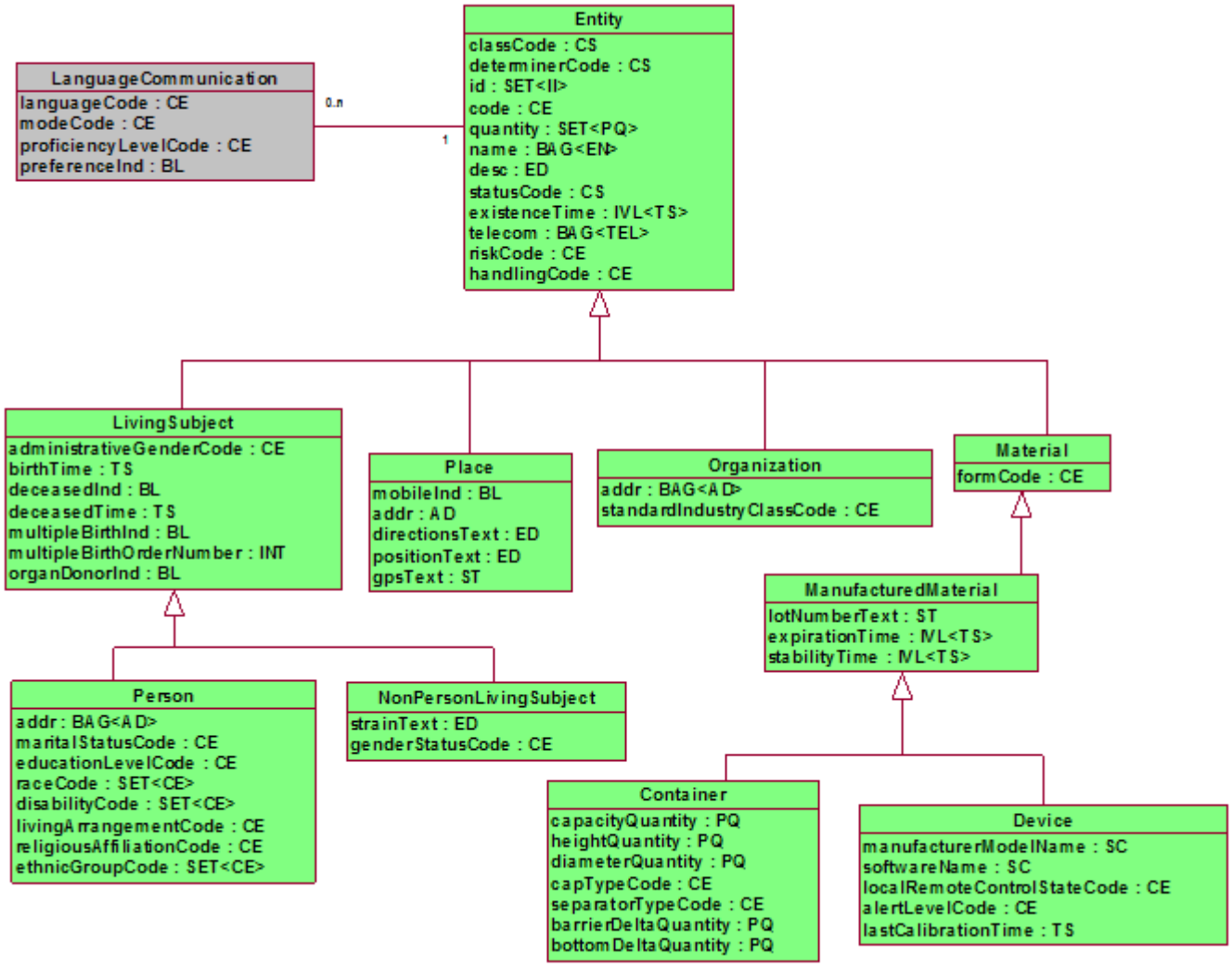
Fundation Classes Subject Area



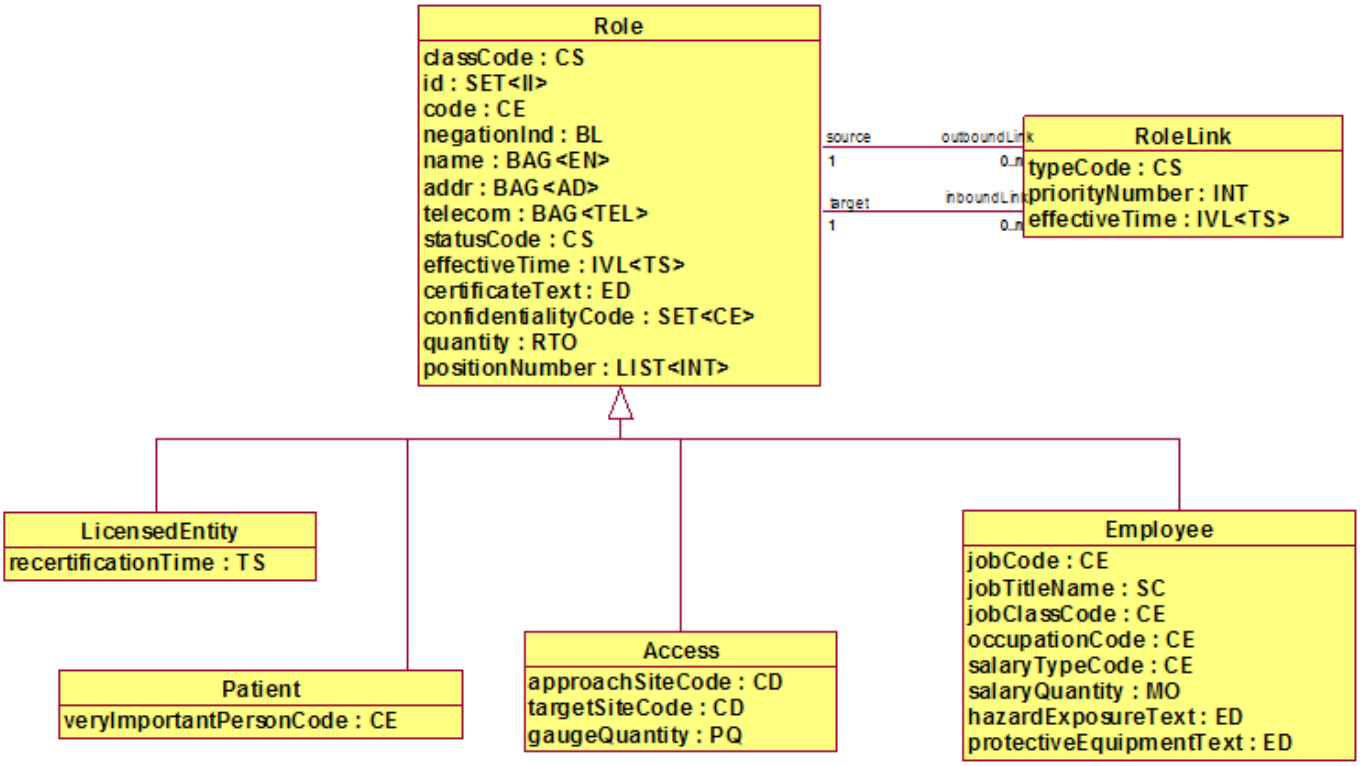
Acts Subject Area



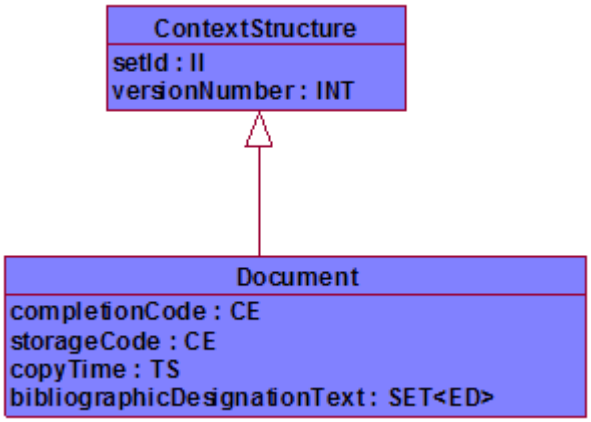
Entities Subject Area



Roles Subject Area



StructuredDocuments Subject Area



Data type

The source files for the data types are hand crafted and contain many specialized methods. These classes also make heavy use of Java Generics (introduced in Java 1.5). Some example data type classes include abstractions such as coded elements (CE), timestamps (TS), physical quantities (PQ), etc.

At current release, we are designated to implement and test following data types:

Name of Type	Type Code
Postal Address	AD
DataValue	ANY
Bag	BAG
Boolean	BL
Concept Descriptor	CD

Coded With Equivalents	CE
Coded Simple Value	CS
Coded Value	CV
Encapsulated Data	ED
Entity Name	EN
General Timing Specification	GTS
Instance Identifier	II
Integer Number	INT
Interval	IVL
Organization Name	ON
Person Name	PN
Physical Quantity	PQ
Ratio	RTO
Character String with Code	SC
Set	SET
Character String	ST
Telecommunication Address	TEL
Trivial Name	TN
Point in Time	TS
Address Part	ADXP
Entity Name Part	ENXP

In addition, following data types are not categorized as core data types from gap analysis. We will, however, support them from implementation perspective, but will not include them in our supported testing level, as compared to the list of data types above.

Name of Type	Type Code
Concept Role	CR
Event-Related Periodic Interval of Time	EIVL
Monetary Amount	MO
ISO Object Identifier	OID
Periodic Interval of Time	PIVL
Physical Quantity Representation	PQR
Unique Identifier String	UID
Universal Resource Locator	URL

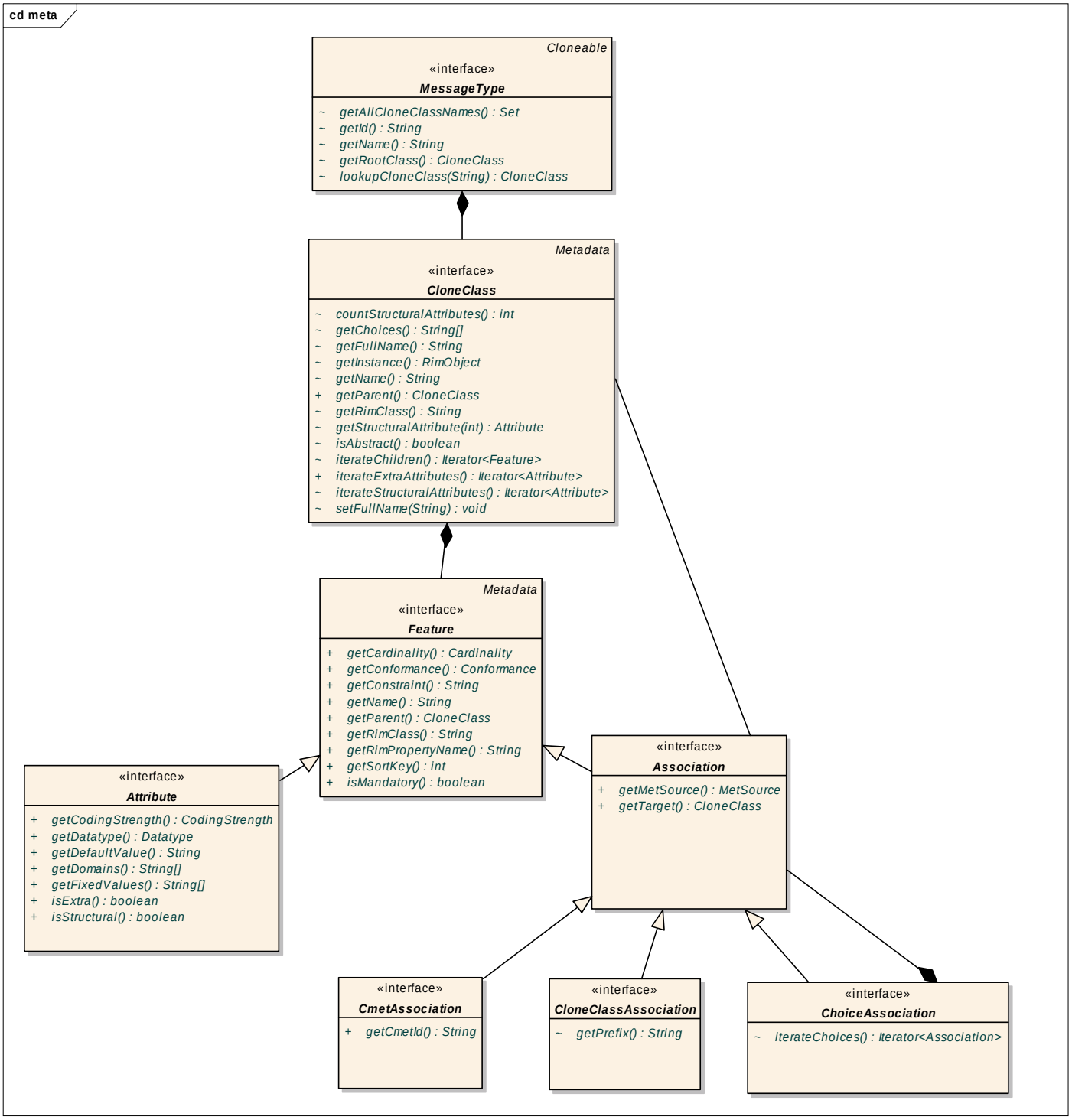
For future reference, following list of data types have not been fully implemented in javasig code base as of 2005 Normative edition and none of them have been referenced on payload classes by RIM 2.07. Therefore, for this release, we will simply exclude consideration of them and focus on those listed above as our main task load.

Name of Type	Type Code
BooleanNonNull	BN
Calendar	CAL
Calendar Cycle	CLCY
Coded Ordinal	CO

GeneratedSequence	GLIST
History	HIST
History Item	HXIT
Non-Parametric Probability Distribution	NPPD
Parametric Probability Distributions over Physical Quantities	PPD<PQ>
Parametric Probability Distributions over Real Numbers	PPD<REAL>
Parametric Probability Distributions over Time Points	PPD<TS>
HL7 Reserved Identifier	RUID
Sampled Sequence	SLIST
DCE Universal Unique Identifier	UUID
Uncertain Value - Probabilistic	UVP

4.2 Meta Data Model

4.2.1 HL7 v3 Meta Data Model

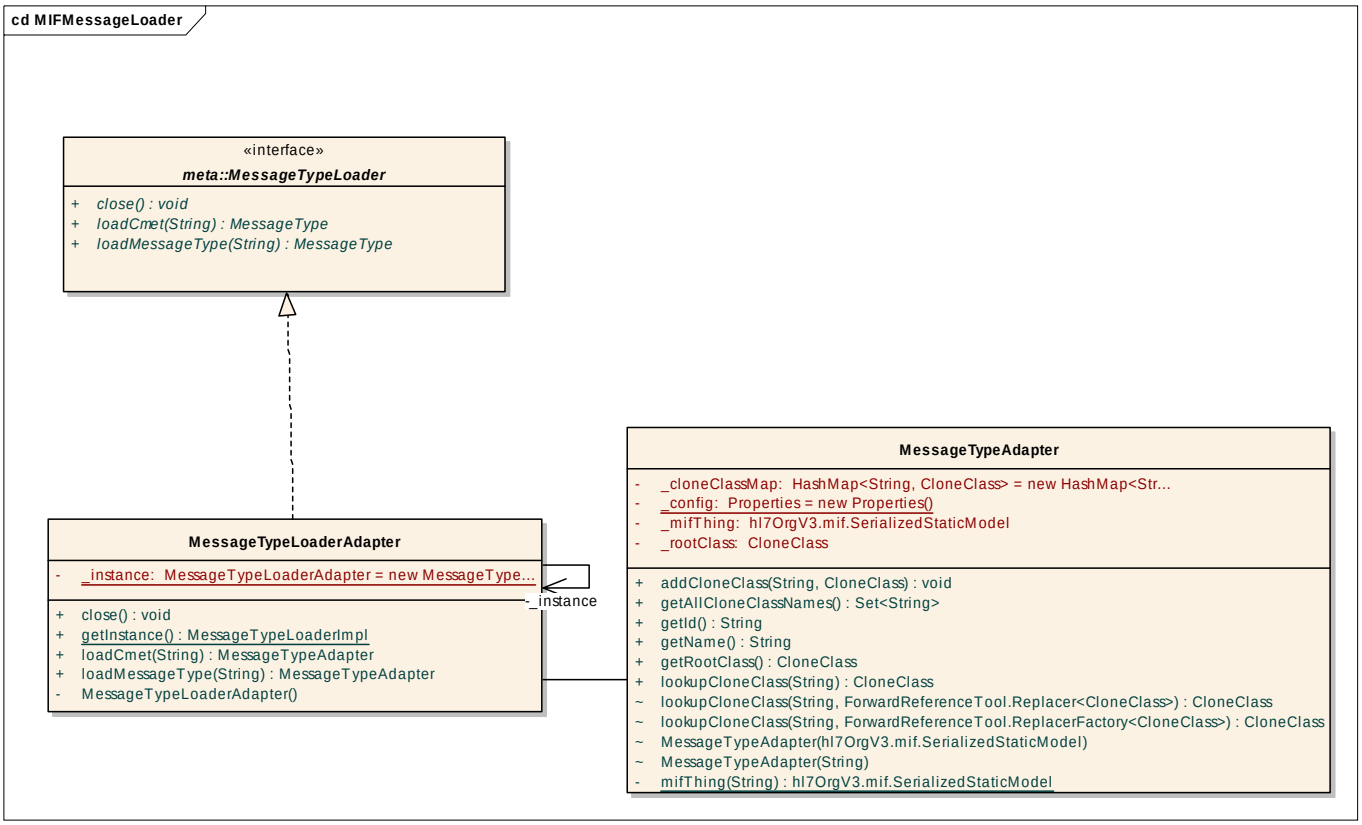


4.2.2 MIF File Loader

HL7 provides two formats Hierarchical Message Definition (HMD) and Model Interchange Format (MIF) for specifying message Meta data (structure, format, and constraints). They are both XML based. HMD is currenty depreciate by HL7. Start caAdapter version 1.3, the system only supports MIF.

The Meta Loader processes the Meta file via a JDOM parser. As key Meta elements are encountered within the input Meta file, Meta classes are created that mirror the attributes and other Meta class associations specified in the file. These Meta classes drive how the RIM object graph is built when messages are actually parsed.

Class MessageTypeLoaderAdpater is MIF file loader, which loads MIF file and converts it into Meta Object Graph.



4.3 Message Parser

The main parsing classes (content handlers) are found in the **org.hl7.xml.parser** package. Presenter classes are located one level higher.

The message parser utilizes a SAX parser when parsing the XM nodes of the input HL7 V3 message. As SAX events fire, data from the SAX event is used to query Meta classes for Meta data. The specific Meta data allows the parser to instantiate a node as a RIM object or HL7 data type object. The Meta data also denotes where to attach the instantiated object. Reflection is used to find the correct “setter” method to “add” the object to the growing RIM graph in memory. When an instance of a RIM class or data type is needed, it is created by a properties based factory.

Since there are many flavors of objects (RIM classes, data types, CMETS or Common Message Element Types) we need specialized content handlers. The specialized content handlers can be dynamically

"switched out" during message parsing. See the DynamicContentHandler class.

Specialized handlers (Presenters) are included for each RIM and data type class. Presenters are called by intermediate content handlers. Examples of these intermediate content handlers include: TreeContentHandler, DataTypeContentHandler, and SimpleTypeContentHandler, and others as well.

As Sax events fire, they are handled by MessageElementContentHandler which in turn can dynamically “suspendWith” to temporarily trade places with a specialized “presenter” class. The final object is handed back down the call stack to a “resultReceiver” with a call to “notifyResult”. Eventually, the returned partial result is added to the growing RIM graph by using reflection to find the correct “setter” method on the parent object.

Once the entire message is parsed, the resulting object graph is the entire in memory representation of the RIM Graph.

4.4 Message Builder

The message building classes are mostly found in the **org.hl7.xml.builder** package. The message builder’s design resembles the parser in that it relies on small specialized content handlers (which in this case are called “builders” instead of “presenters”).

If you look inside a presenter class you will find an inner class of type “builder”. This design choice was made so that once a presenter is written it is able to handle both message parsing and message building.

The message builder uses a clever “trick” to generate the XML output. Instead of writing many print statements, another approach was taken utilizing the "Identify transform" feature of XSLT transforms.

The “Identity transform” is a special case where the incoming XML message is sent through the XSLT transform machinery, but no actual transform is done. The newly generated output is exactly the same as the XML input: but the output isn't simply copied. The input generated SAX events and these events were used to create the XML output. Since no transform is performed, the resulting message is the same as the input message.

If you could read or parse the in memory RIM graph and pretend to be a SAX XMLReader, essentially generating the same events as if you were reading an XML stream, then you could use that reader in an identity transform to create XML output. That is the approach that was taken in the message builder (credit: Gunther Schadow).

The mechanism that masquerades as a SAX XML reader is made up of the classes XMLSpeaker and XMLRimGraphSpeaker. As the SAX events are emitted by “parsing” the in memory RIM graph, they are handled by “MessageBuilders”. The message builders are then dynamically switched for one another as appropriate. Finally one of the builders emits a SAX event to an internal member of type “org.xml.sax.ContentHandler”. This member happens to be the "_contentHandler" member from the abstract class XMLSpeaker. Once this event fires, the XSLT transform takes over and the XML for that particular node is created.

Human Computer Interface on page 37) that allows end-users the ability to perform commands and functions in an intuitive and easy to use computer interface.

Topics in this chapter include:

4.5 GUI Overview

The GUI subsystem presents an interactive graphical user interface that provides users the ability to define, modify, export and import specification that expresses a source file format and to validate a source data file against a specification definition. It also allows users to use a drag-and-drop on the user interface to map between source and target specifications and consequently realize a mapping specification. This subsystem is designed using standard user interface techniques and follows the NCICB GUI design guidelines. The GUI class diagram is shown in *Figure 4-1* and described in sections starting with *4.6 MainFrame* on page 26.

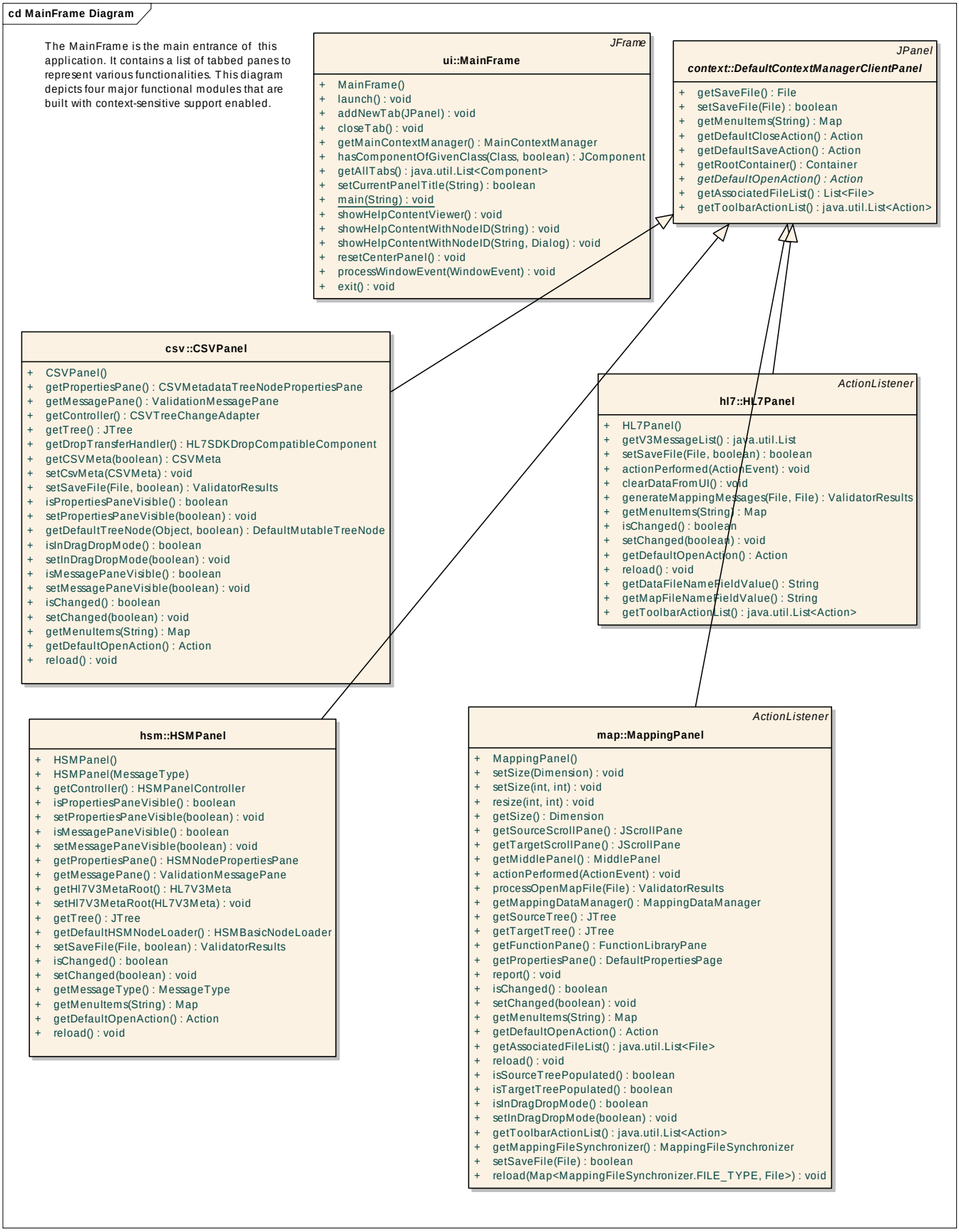

cd MainFrame Diagram
The MainFrame is the main entrance of this application. It contains a list of tabbed panes to represent various functionalities. This diagram depicts four major functional modules that are built with context-sensitive support enabled.
JFrame
ui::MainFrame
+ MainFrame()
+ launch() : void
+ addNewTab(JPanel) : void
+ closeTab() : void
+ getMainContextManager() : MainContextManager
+ hasComponentOfGivenClass(Class, boolean) : JComponent
+ getAllTabs() : java.util.List<Component>
+ setCurrentPanelTitle(String) : boolean
+ main(String) : void
+ showHelpContentViewer() : void
+ showHelpContentWithNodeID(String) : void
+ showHelpContentWithNodeID(String, Dialog) : void
+ resetCenterPanel() : void
+ processWindowEvent(WindowEvent) : void
+ exit() : void
JPanel
context::DefaultContextManagerClientPanel
+ getSaveFile() : File
+ setSaveFile(File) : boolean
+ getMenuItems(String) : Map
+ getDefaultCloseAction() : Action
+ getDefaultSaveAction() : Action
+ getRootContainer() : Container
+ getDefaultOpenAction() : Action
+ getAssociatedFileList() : List<File>
+ getToolbarActionList() : java.util.List<Action>
csv::CSVPanel
+ CSVPanel()
+ getPropertiesPane() : CSVMetadataTreeNodePropertiesPane
+ getMessagePane() : ValidationMessagePane
+ getController() : CSVTreeChangeAdapter
+ getTree() : JTree
+ getDropTransferHandler() : HL7SDKDropCompatibleComponent
+ getCSVMeta(boolean) : CSVMeta
+ setCsvMeta(CSVMeta) : void
+ setSaveFile(File, boolean) : ValidatorResults
+ isPropertiesPaneVisible() : boolean
+ setPropertiesPaneVisible(boolean) : void
+ getDefaultTreeNode(Object, boolean) : DefaultMutableTreeNode
+ isInDragDropMode() : boolean
+ setInDragDropMode(boolean) : void
+ isMessagePaneVisible() : boolean
+ setMessagePaneVisible(boolean) : void
+ isChanged() : boolean
+ setChanged(boolean) : void
+ getMenuItems(String) : Map
+ getDefaultOpenAction() : Action
+ reload() : void
ActionListener
hl7::HL7Panel
+ HL7Panel()
+ getV3MessageList() : java.util.List
+ setSaveFile(File, boolean) : boolean
+ actionPerformed(ActionEvent) : void
+ clearDataFromUI() : void
+ generateMappingMessages(File, File) : ValidatorResults
+ getMenuItems(String) : Map
+ isChanged() : boolean
+ setChanged(boolean) : void
+ getDefaultOpenAction() : Action
+ reload() : void
+ getDataFileNameFieldValue() : String
+ getMapFileNameFieldValue() : String
+ getToolbarActionList() : java.util.List<Action>
hsm::HSMPanel
+ HSMPanel()
+ HSMPanel(MessageType)
+ getController() : HSMPanelController
+ isPropertiesPaneVisible() : boolean
+ setPropertiesPaneVisible(boolean) : void
+ isMessagePaneVisible() : boolean
+ setMessagePaneVisible(boolean) : void
+ getPropertiesPane() : HSMNodePropertiesPane
+ getMessagePane() : ValidationMessagePane
+ getHl7V3MetaRoot() : HL7V3Meta
+ setHl7V3MetaRoot(HL7V3Meta) : void
+ getTree() : JTree
+ getDefaultHSMNodeLoader() : HSMBasicNodeLoader
+ setSaveFile(File, boolean) : ValidatorResults
+ isChanged() : boolean
+ setChanged(boolean) : void
+ getMessageType() : MessageType
+ getMenuItems(String) : Map
+ getDefaultOpenAction() : Action
+ reload() : void
ActionListener
map::MappingPanel
+ MappingPanel()
+ setSize(Dimension) : void
+ setSize(int, int) : void
+ resize(int, int) : void
+ getSize() : Dimension
+ getSourceScrollPane() : JScrollPane
+ getTargetScrollPane() : JScrollPane
+ getMiddlePanel() : MiddlePanel
+ actionPerformed(ActionEvent) : void
+ processOpenMapFile(File) : ValidatorResults
+ getMappingDataManager() : MappingDataManager
+ getSourceTree() : JTree
+ getTargetTree() : JTree
+ getFunctionPane() : FunctionLibraryPane
+ getPropertiesPane() : DefaultPropertiesPage
+ report() : void
+ isChanged() : boolean
+ setChanged(boolean) : void
+ getMenuItems(String) : Map
+ getDefaultOpenAction() : Action
+ getAssociatedFileList() : java.util.List<File>
+ reload() : void
+ isSourceTreePopulated() : boolean
+ isTargetTreePopulated() : boolean
+ isInDragDropMode() : boolean
+ setInDragDropMode(boolean) : void
+ getToolbarActionList() : java.util.List<Action>
+ getMappingFileSynchronizer() : MappingFileSynchronizer
+ setSaveFile(File) : boolean
+ reload(Map<MappingFileSynchronizer.FILE_TYPE, File>) : void

Figure 4-1 GUI class framework class diagram

4.6 MainFrame

`MainFrame` is the root object of the GUI. It exposes functionality for adding a list of tabbed panels. The panels are described in four classes as shown in *Figure 4-1*:

1. `MappingPanel – The` MappingPanel class exposes functionality for retrieving a set of mappings and GUI objects that visually illustrate the mappings between panels. Two classes extend the `MappingPanel (`both of which are Java Swing objects):
    a. `SourceTree` - The SourceTree class builds a hierarchical tree structure that represents a source data structure (for example, CSV specification).
    b. `TargetTree` - The TargetTree class builds a hierarchical tree structure that represents a target data structure (for example, HL7 HMD).
2. `CSVPanel` – The CSVPanel class exposes functionality for loading and displaying a Java Swing Panel that stores CSV data.
3. `HL7Panel` – The HL7Panel class exposes functionality for loading and displaying a Java Swing Panel that generates and presents HL7 v3 Messages.
4. `HSMPanel` – The HSMPanel class exposes functionality for loading and displaying a Java Swing Panel that helps user update the HL7 v3 specification.

4.7 Context-Sensitive Framework

The caAdapter application is aimed to support multiple file formats, such as CSV specification, mapping specification, etc. each of which has various sets of functionalities to facilitate users from navigations to conducting analysis work. To maintain consistent look and feel and accommondate individual requirement as well as future extensibility, the GUI needs to be designed and implemented to support context-sensitive menu and tool bar navigations. On the menu bar, only the menu options available for your current display appear in black font; the other options are grayed-out. On the tool bar, only the options that are available for your current display are presented.

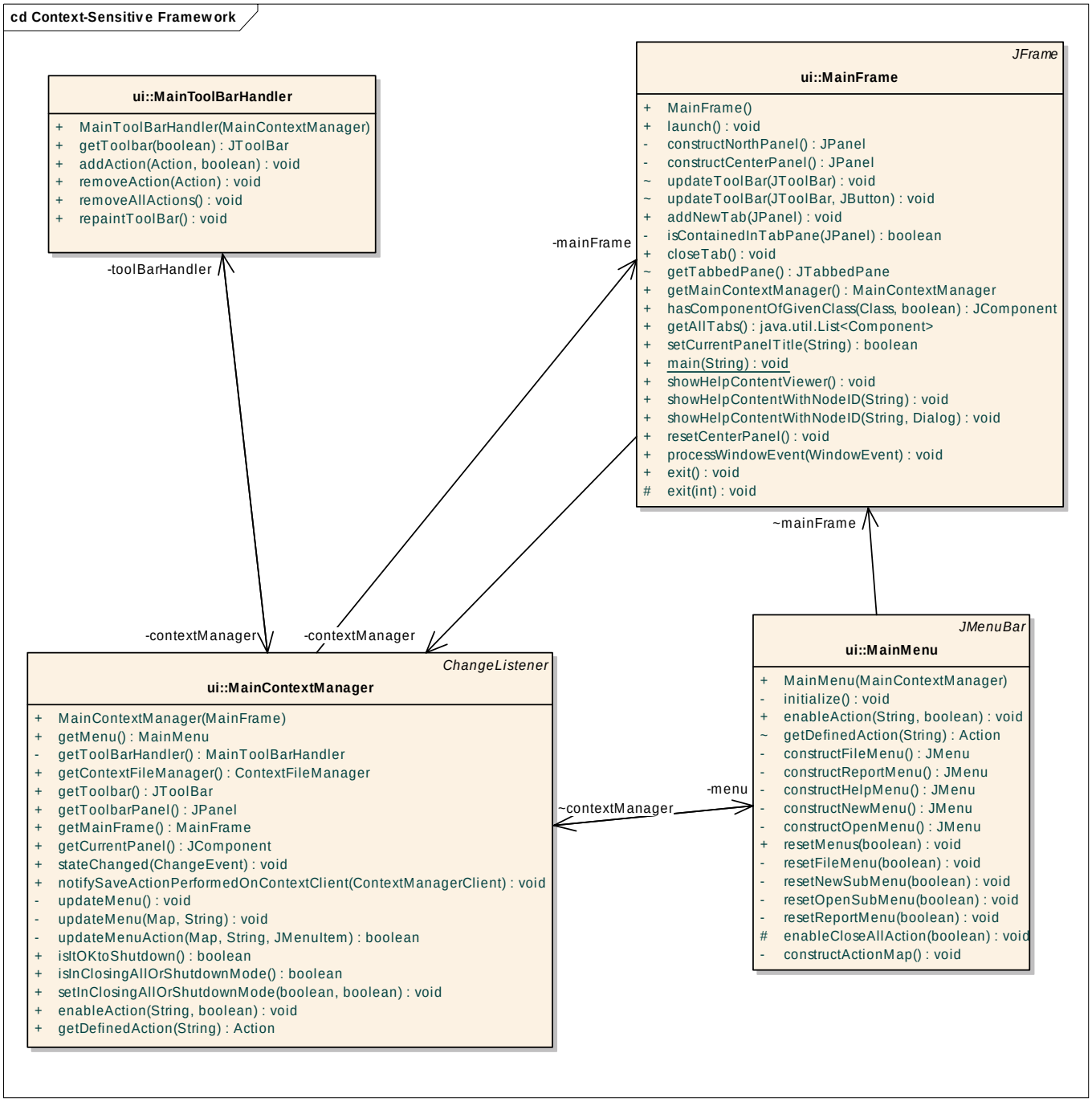


Figure 4-2 context-sensitive high-level design

4.8 NodeLoaders

The Figure 4-3 Node Loader class diagram illustrates the NodeLoader class diagram that exposes functionality for converting business objects into something the aforementioned GUI can understand.

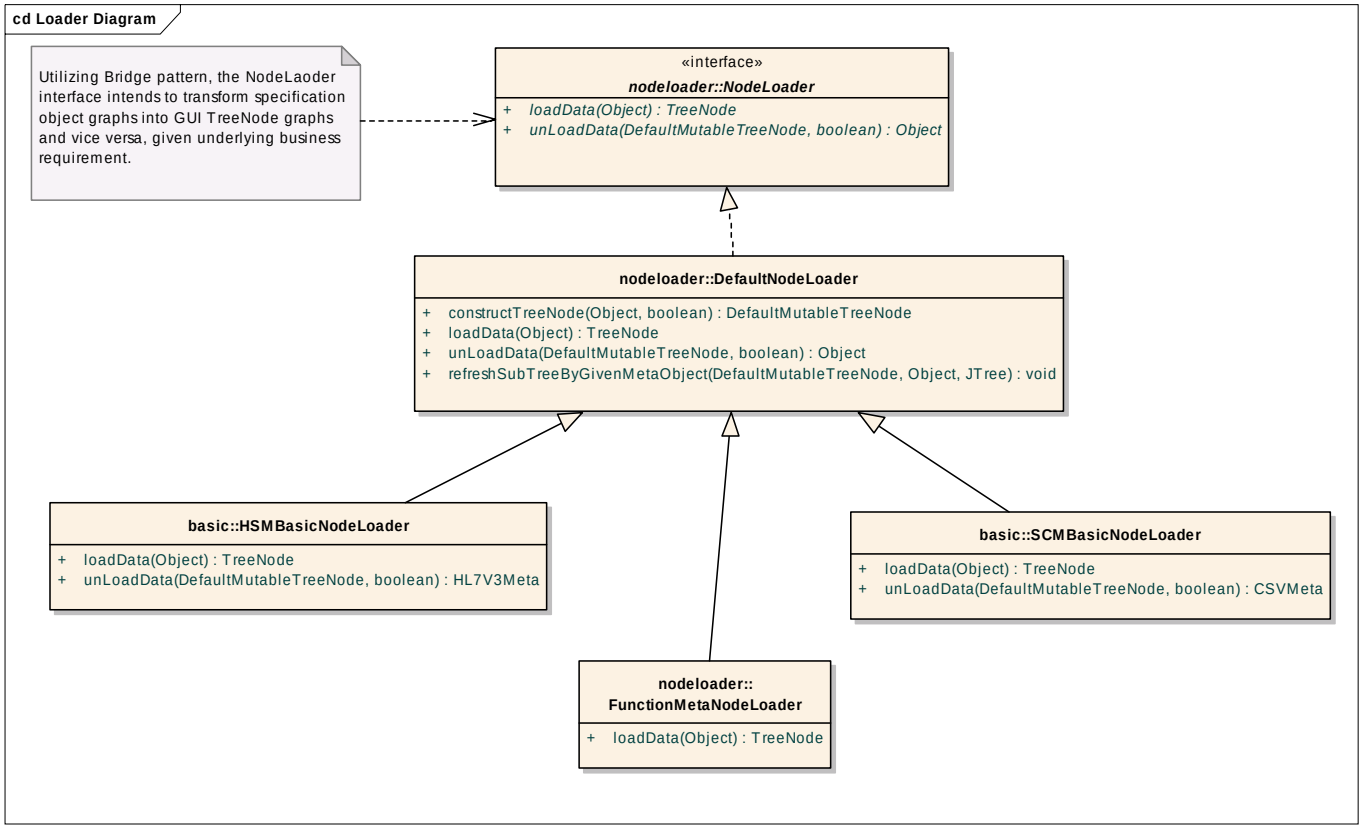


Figure 4-3 Node Loader class diagram

The NodeLoader and its descendant classes help bridge the gap between the presentation and middle (that is, business logic) tier. The NodeLoader interface is intended to convert object graphs into a GUI TreeNode graph and vice versa. The `HL7MetaNodeLoader` class, for example, takes a Message Typeid as input and contains functionality for loading and processing nodes in a graphical tree structure. The `CSVNodeLoader` class takes a CSVMeta object as input and exposes functionality for loading, unloading and processing segment data related to CSV data structures. Both `HL7MetaNodeLoader` and `CSVNodeLoader` classes extend the `NodeLoader` interface class that exposes a method for loading data. This design supports extensibility (as defined under the design principles) so that other loaders may be added and called by the `NodeLoader` interface class.

5. CSV Subsystem

The CSV Subsystem contains the ability to create and process CSV specifications and the ability to parse (and validate) CSV data against a particular CSV specification. For this release, we will focus on segmented CSV files. Segmented CSV files are defined as a comma delimited sequence of fields in which hierarchy can be derived based on a prefixed identifier (in the first column). Each row is considered a segment.

5.1 CSV Specifications

CSV Specification describes the structure of a CSV instance, and hence defines the format for the CSV file content. In essence, it is a CSV specification in the same way an XML schema definition (XSD) is a specification of an XML instance.

5.1.1 CSV Specification Object Graph

Figure 5-1 illustrates a class diagram and the relationships between classes involved in setting and retrieving information about a CSV specification.

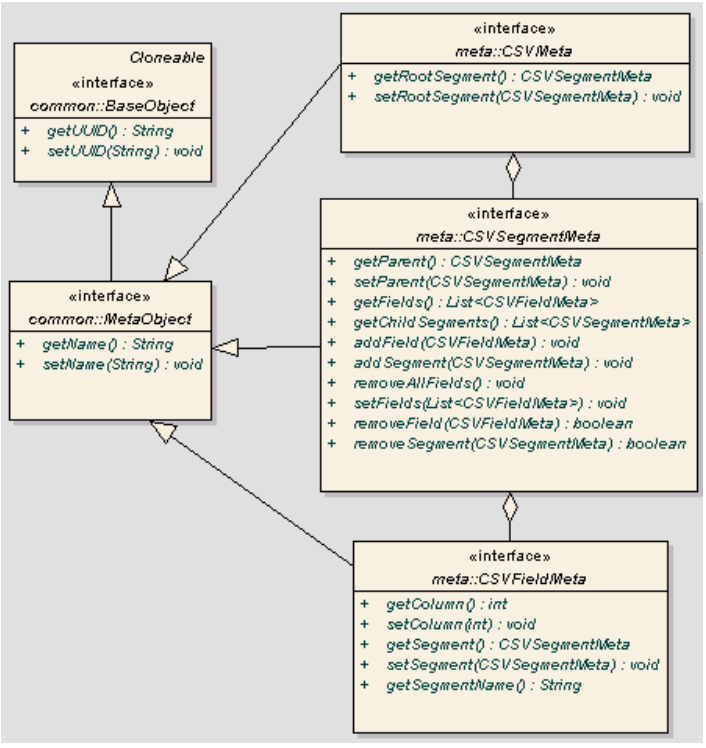


Figure 5-1 CSV Specification object graph class diagram

5.2 Segmented CSV Data

A segmented CSV data file is an ASCII file that consists of series of comma delimited fields that are prefixed with a specified delimiter, or segment name, in the first column.

5.2.1 CSV Data Object Graph

Figure 5-2 illustrates an in-memory representation of a segmented CSV data object graph.

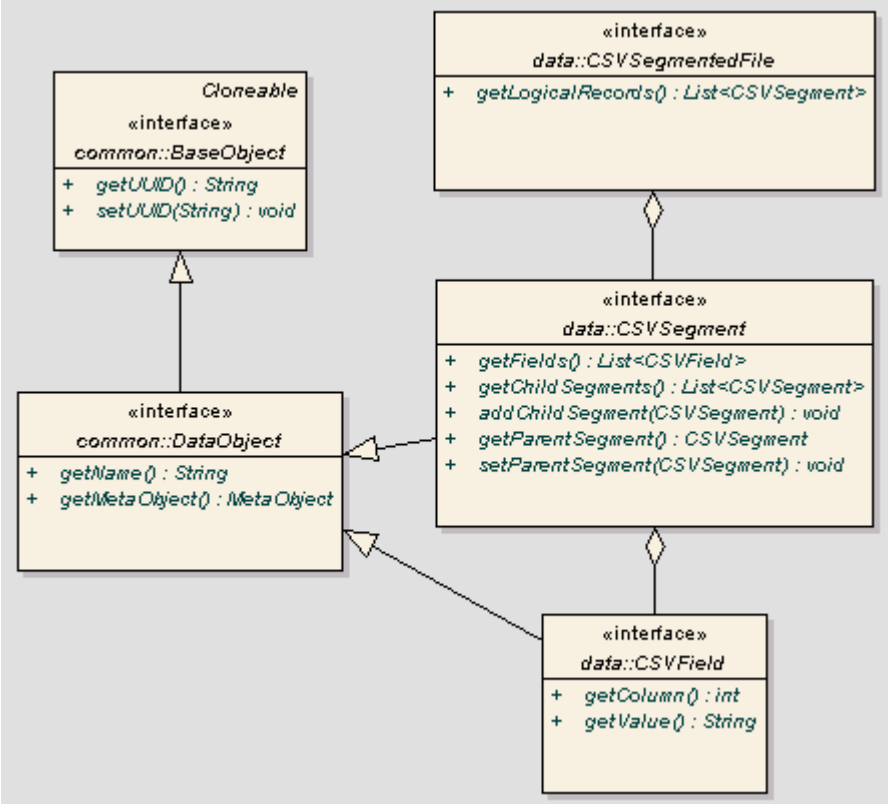


Figure 5-2 In-Memory Representation of CSV object graph class diagram

A CSV object graph is created based on two inputs:

1. a CSV instance
2. a CSV specification

In *Figure 5-2*, there are three main interface classes:

1. `CSVSegmentedFile` - The `CSVSegmentedFile` class exposes functionality for retrieving a set of logical CSV records.
2. `CSVSegment` - The `CSVSegment` class exposes functionality for retrieving/changing a segment's parent segment, child segments and fields.
3. `CSVField` - The `CSVField` class exposes functionality for retrieving/changing a field's value.

5.3 Builders and Parsers

Figure 5-3 illustrates a sequence diagram with a set of objects and messages for parsing a segmented CSV data file based on CSV specification.

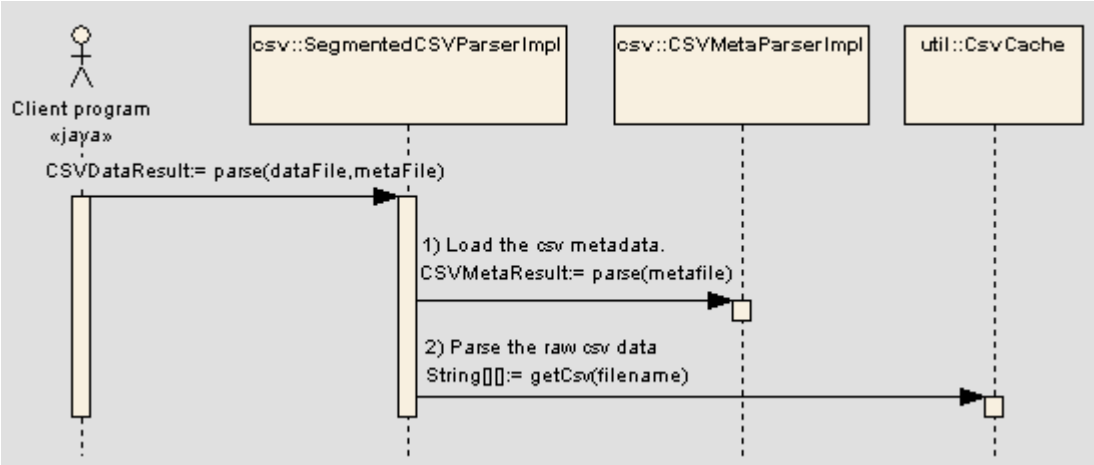


Figure 5-3 CSV Specification and Data Parsers

The primary classes in *Figure 5-3* are:

1. `SegmentedCSVParserImpl` - The `SegmentedCSVParserImpl` parses a segmented CSV file based on a CSV specification and generates a CSV data object graph accordingly.
2. `CSVMetaParserImpl` - The `CSVMetaParserImpl` parses a CSV specification into a CSV specification object graph.
3. `CsvCache` – The `CsvCache` parses a raw CSV datafile into a two dimetional String array.

In addition, not included in the sequence diagram, there is:

`CSVMetaBuilderImpl` - The `CSVMetaBuilderImpl` builds a CSV specification file from a CSV specification object graph.

6. Mapping Subsystem

The Mapping Subsystem contains the ability to create and process mapping files. These mapping files define the relationship between two different specifications. They are persisted as XML.

6.1 Mapping Classes

Shown in *Figure 6-1* is the mapping objects class diagram. This is the in memory representation of all mapping information and it is the input for the map parser, the map builder and the map processor.

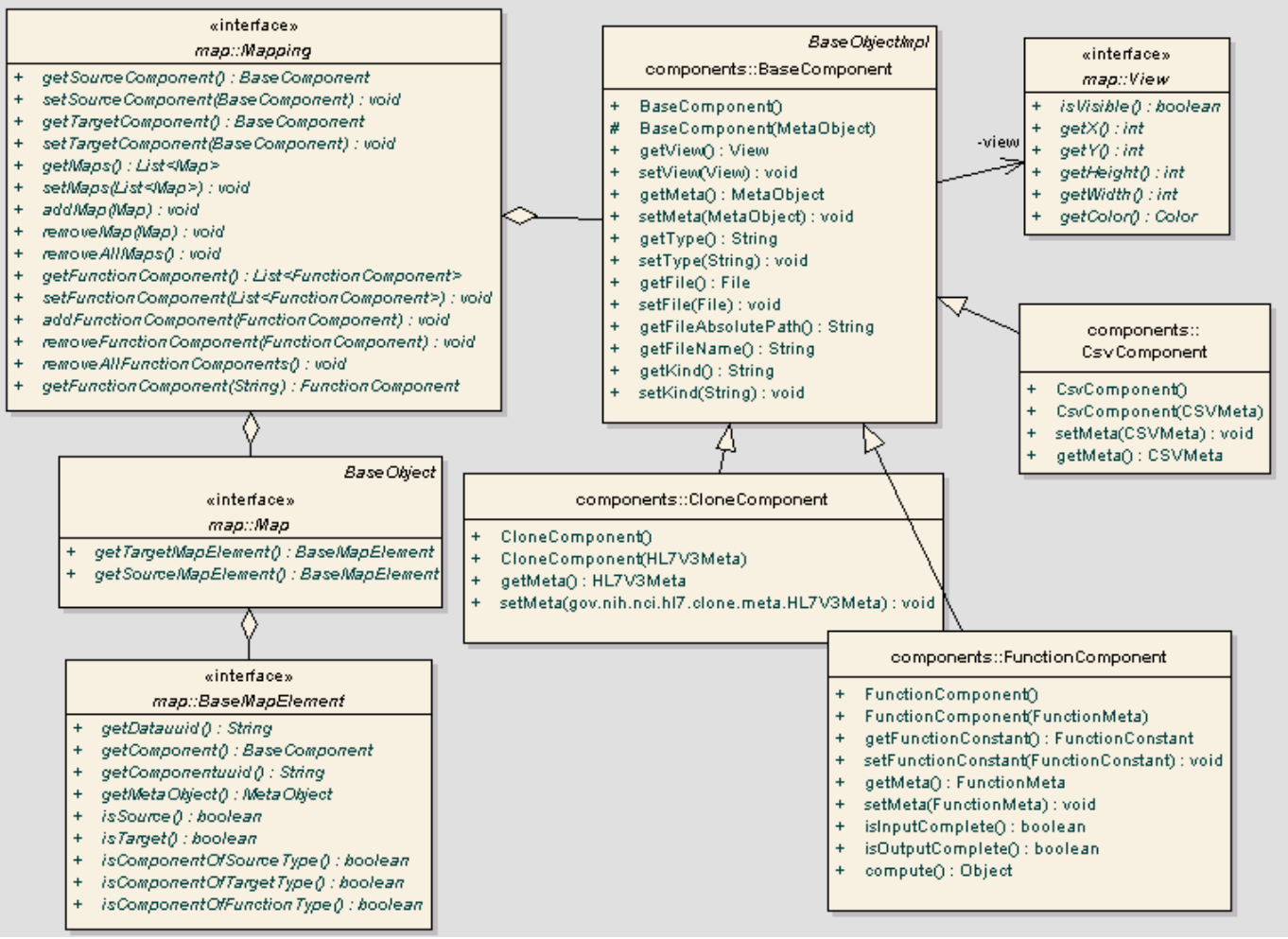


Figure 6-1 Mapping Objects class diagram

6.2 Mapping Processor

The Mapping Processor generates Clone Data objects based on two inputs: 1) The segmented CSV data and 2) the mapping objects. This is where all the mapping rules are defined.

6.3 RIM Graph Generator

The RIM Graph Generator generates a RIM-based object graph based on Clone Data objects (Clone Data objects are created by the MapProcessor).

7. Function Subsystem

The Function Subsystem provides the ability to serve up function metadata (which can be persisted within a mapping file) and execute its underlying implementation (by the map processor). The system presents an easy to use API and access to functions, function groups and function parameters.

7.1 Function Objects

Figure 7-1 depicts classes responsible for carrying out operations to facilitate data transformation functionality for the Function feature.

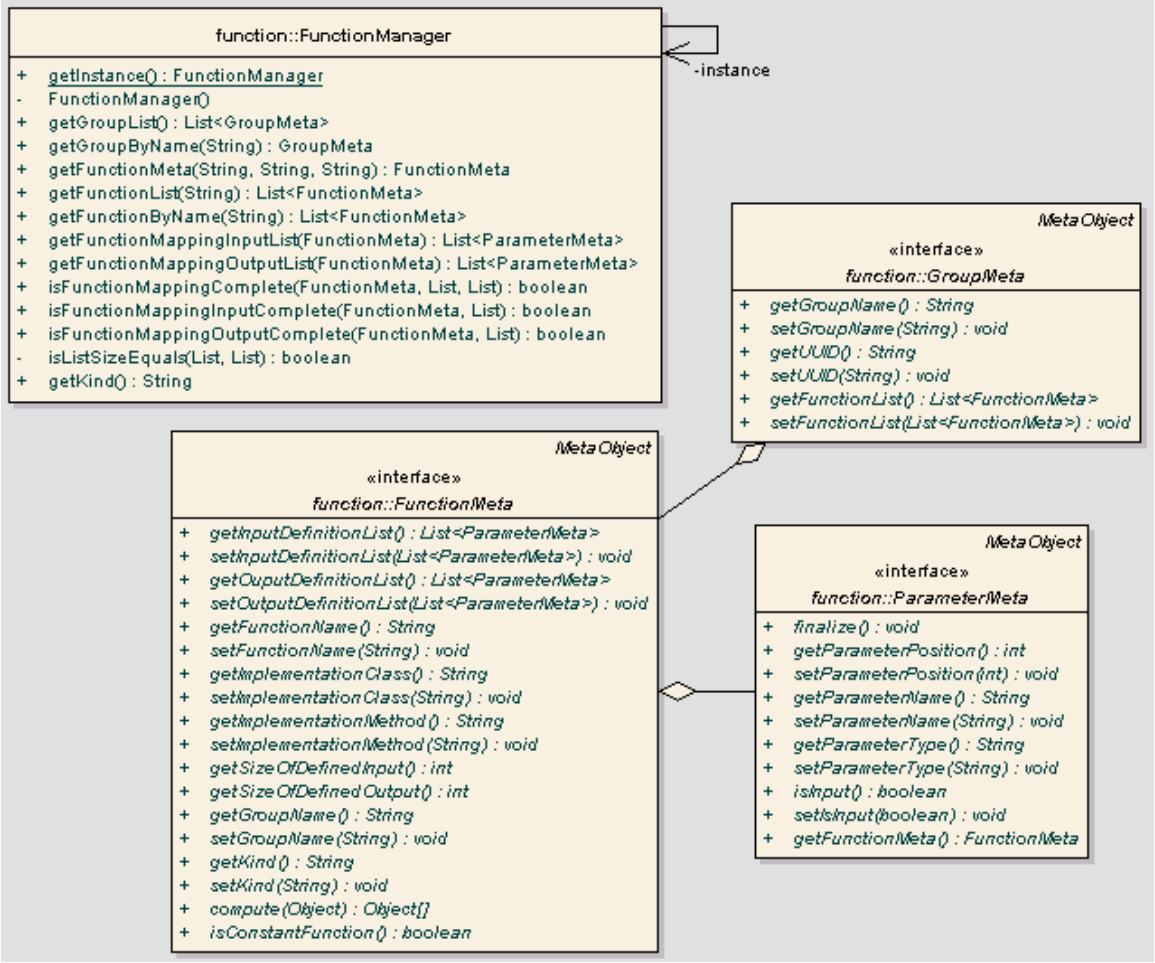


Figure 7-1 Function Meta Subsystem class diagram

8. HL7 v3 Subsystem

The HL7 v3 subsystem provides the ability to specialize the HL7 v3 specification based on a particular user's need. For example, a user can subclass datatypes, select clone choices and apply default values.

8.1 HL7 v3 Specification

Figure 8-1 depicts a set of clone meta object classes which represent a specialized HL7 v3 message. This structure can be built and parsed as an XML document (h3s file).

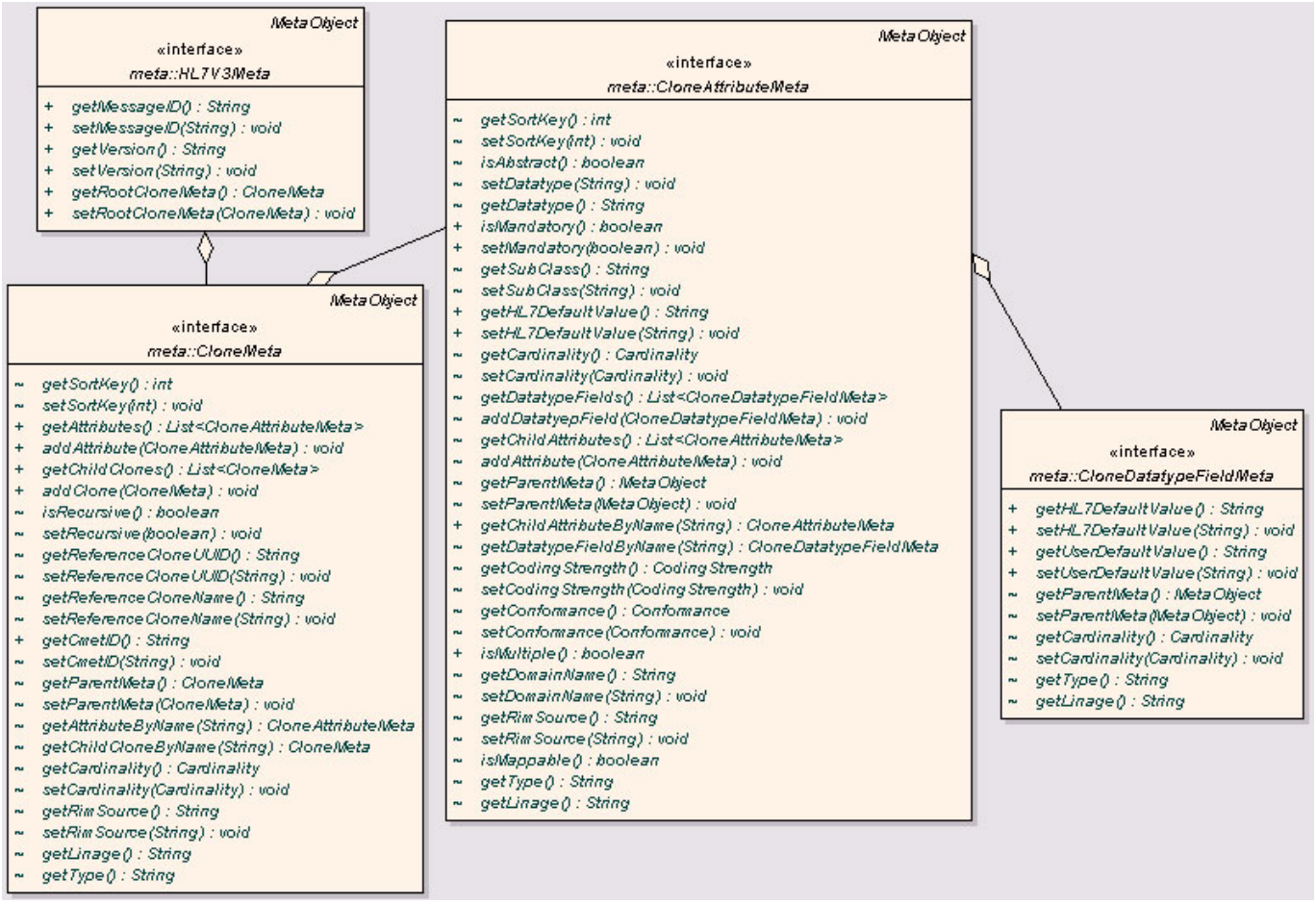


Figure 8-1 HL7 specification class diagram

8.2 HL7 v3 Data Objects

Figure 8-2 depicts a set of clone data object classes which is used to generate RIM objects from.

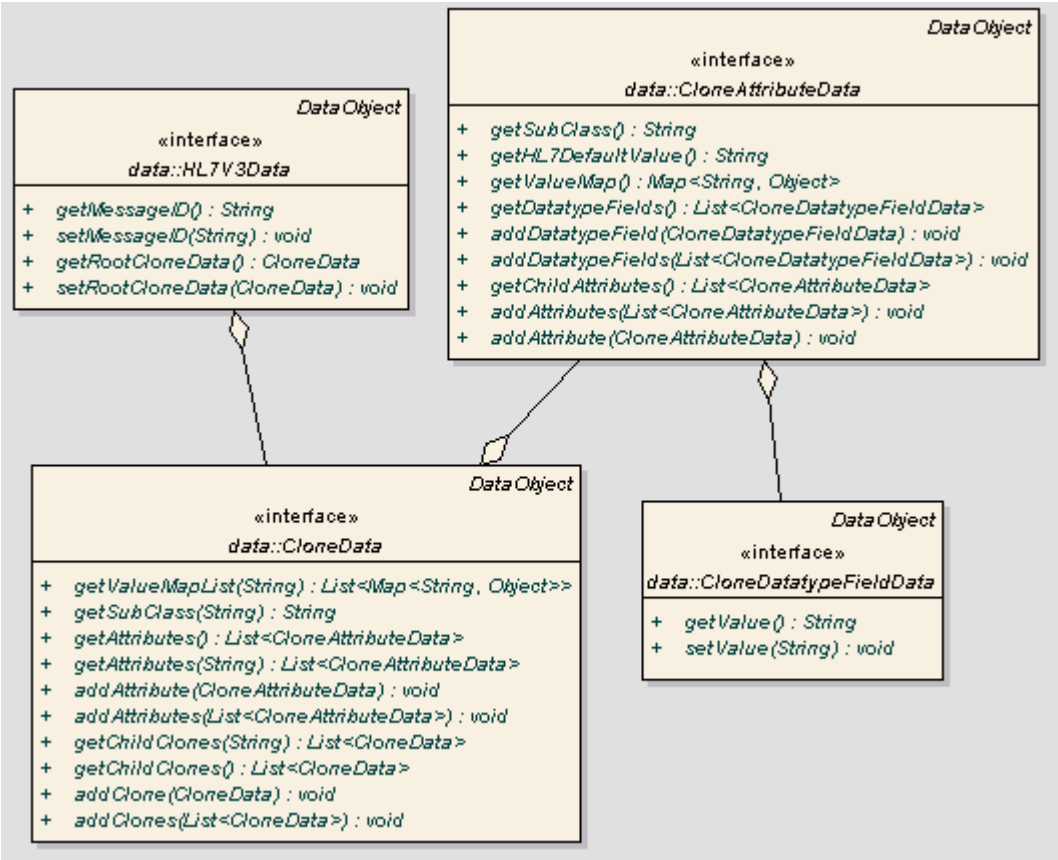


Figure 8-2 HL7 Data class diagram

9. HL7 v3 API

The purpose of this chapter is to present various aspects of knowledge related to HL7 v3 API that caAdapter tool depends upon.

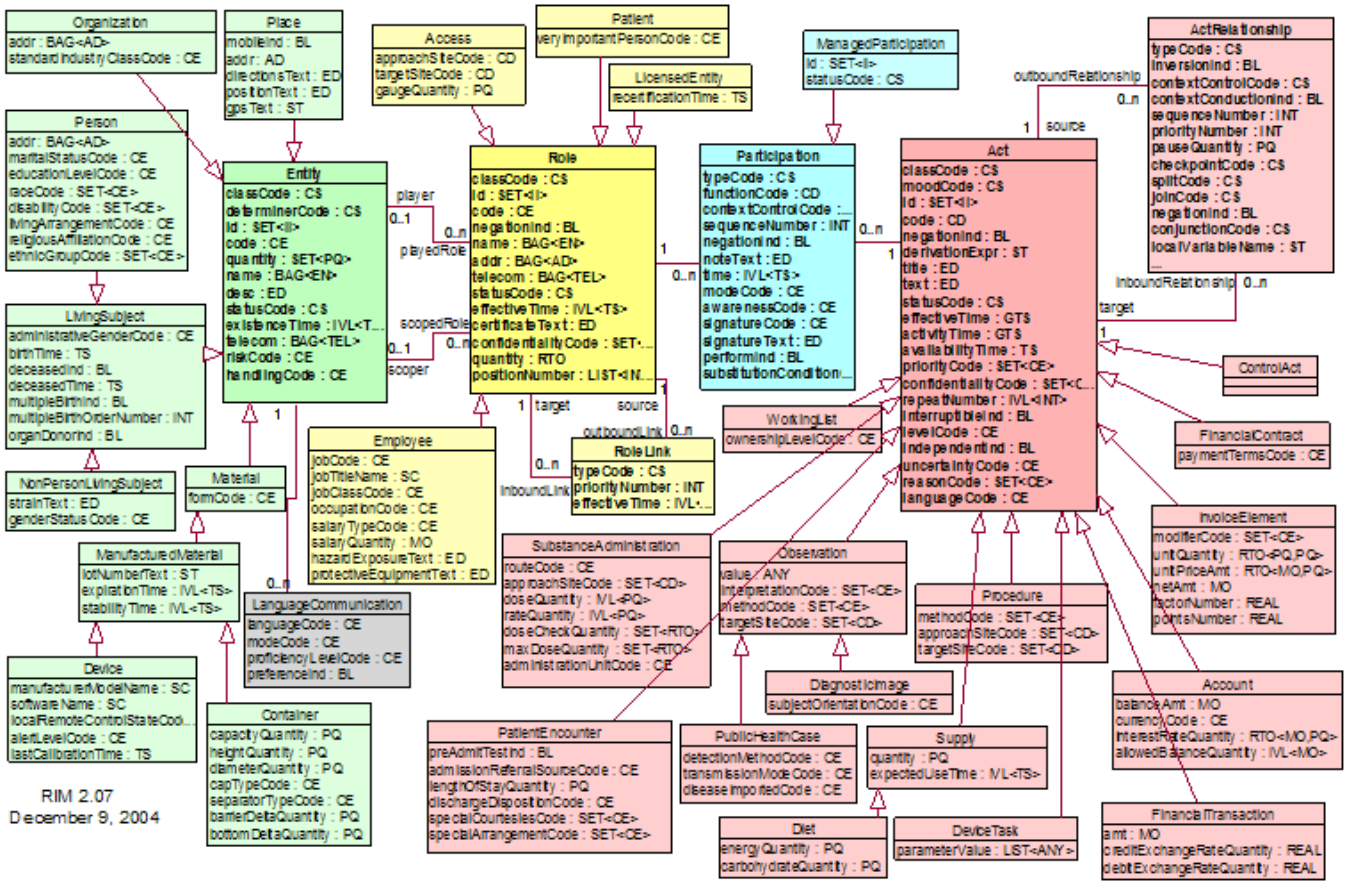
Topics in this chapter include:

9.1 Reference Information Model

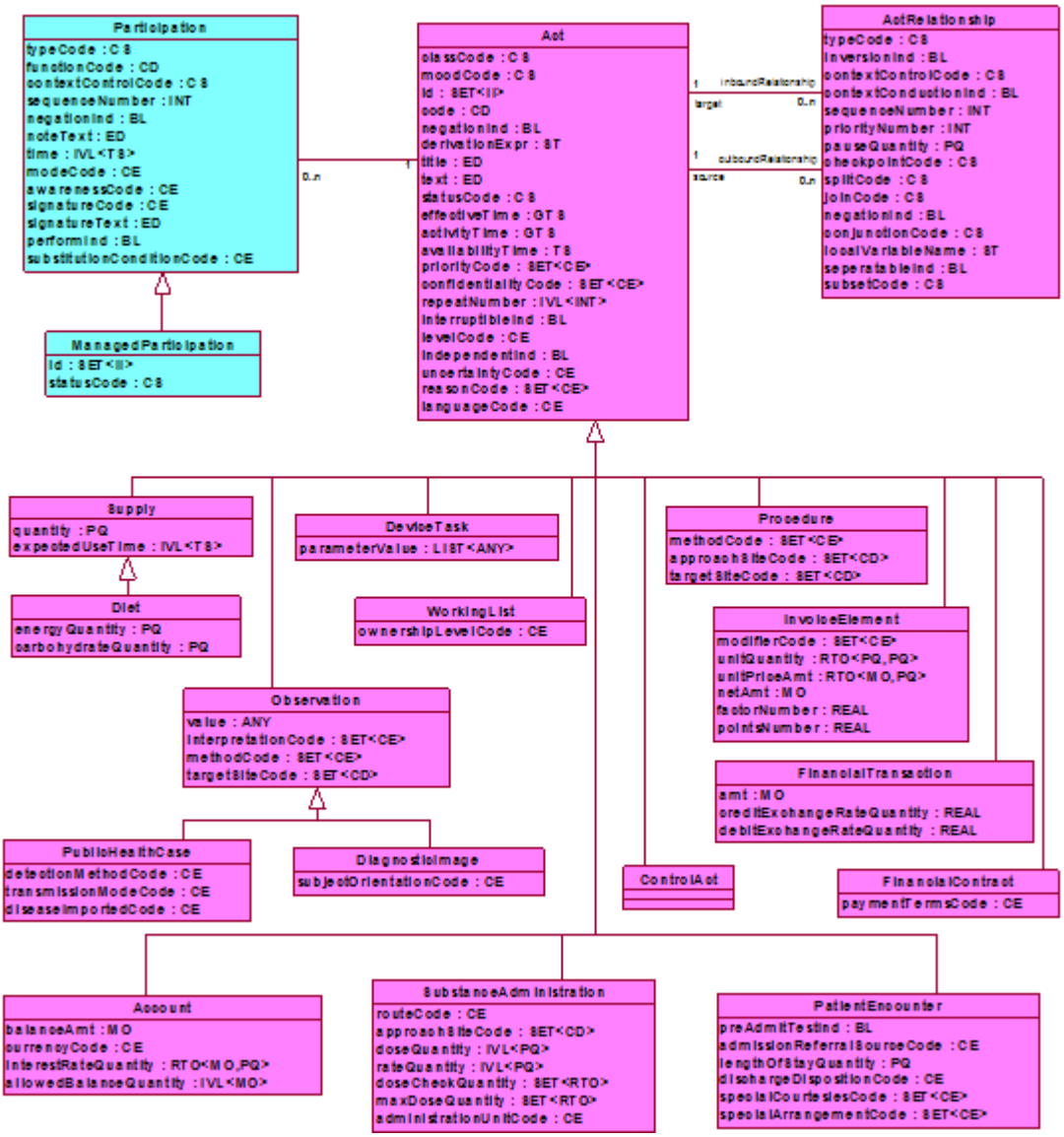
The source files for these classes are automatically generated from the XSLT transform of definition files. The RIM classes are based on high level abstractions such as patients, observations, procedures, roles, acts, etc.

At current release, the team is responsible to migrate the current support of RIM 2.02 to the latest Normative release as of RIM 2.07. A comparative research has been conducted to collect the gap analyis on accumulative variations since RIM 2.02 till RIM 2.07. A separate spreadsheet document has been constructed to denote corresponding changes in several areas that are covered below.

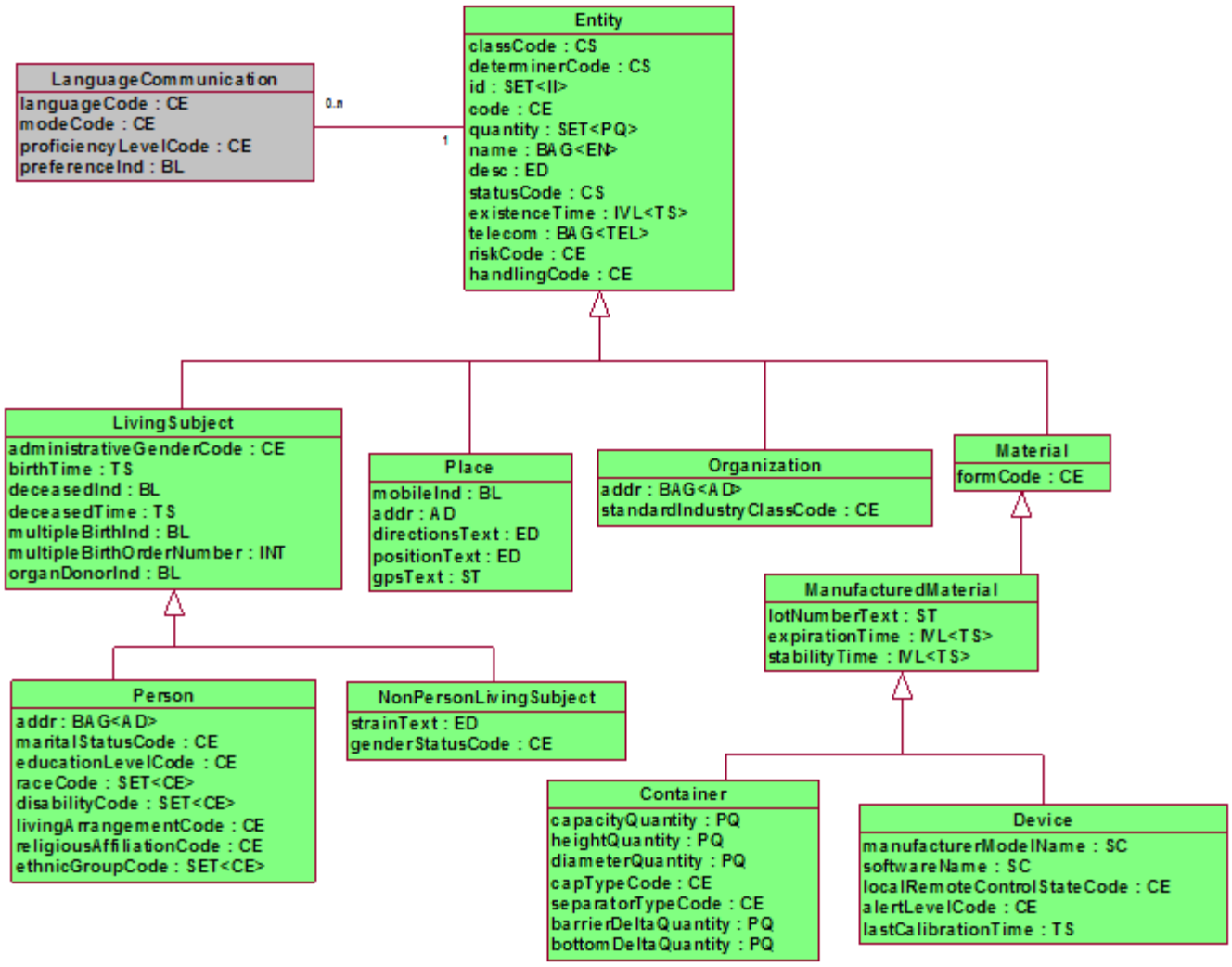
9.1.1 Fundation Classes Subject Area



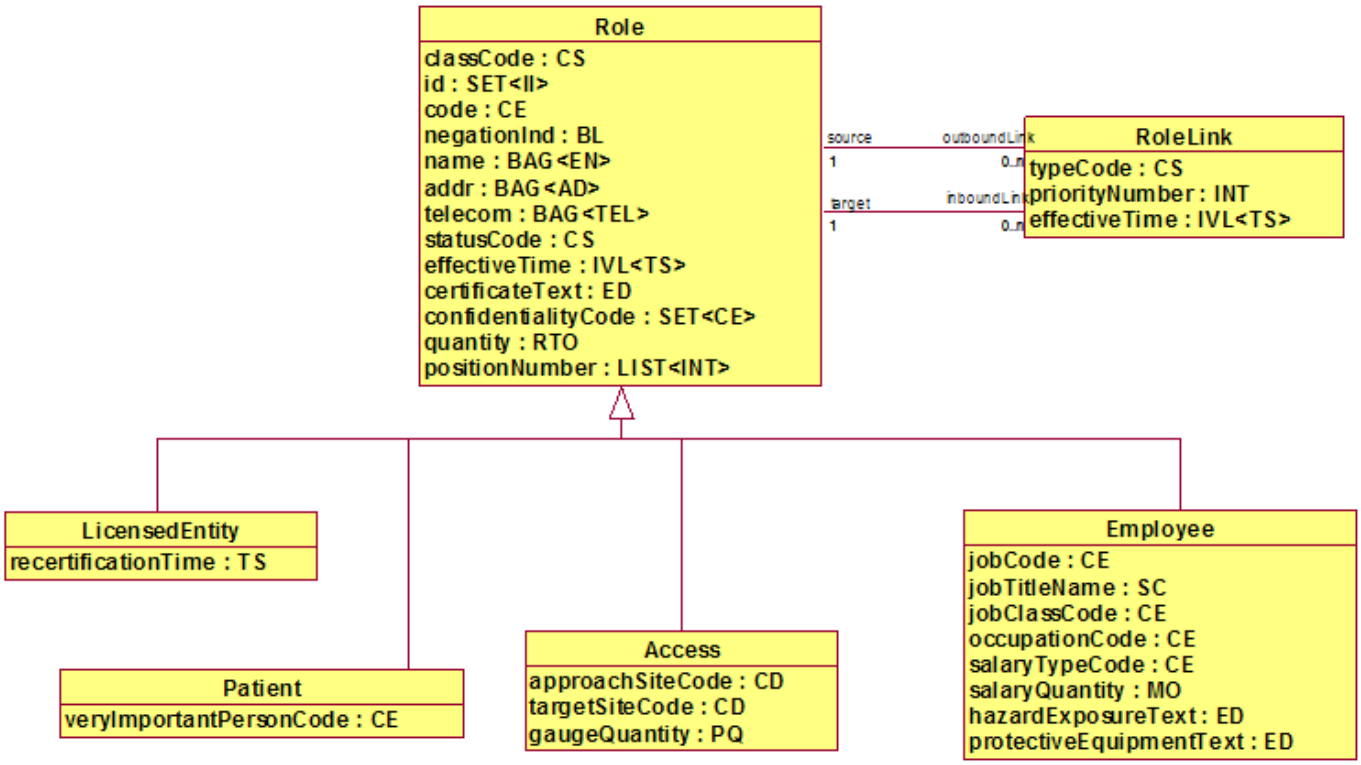
9.1.2 Acts Subject Area



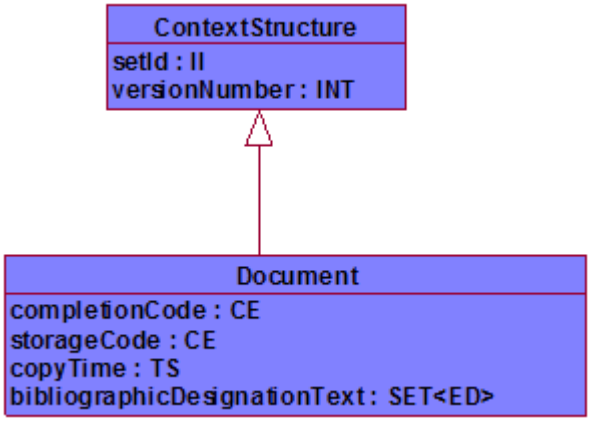
9.1.3 Entities Subject Area



9.1.4 Roles Subject Area



9.1.5 StructuredDocuments Subject Area



9.2 Data type

The source files for the data types are hand crafted and contain many specialized methods. These classes also make heavy use of Java Generics (introduced in Java 1.5). Some example data type classes include abstractions such as coded elements (CE), timestamps (TS), physical quantities (PQ), etc.

At current release, we are designated to implement and test following data types:

Name of Type	Type Code

Postal Address	AD
DataValue	ANY
Bag	BAG
Boolean	BL
Concept Descriptor	CD
Coded With Equivalents	CE
Coded Simple Value	CS
Coded Value	CV
Encapsulated Data	ED
Entity Name	EN
General Timing Specification	GTS
Instance Identifier	II
Integer Number	INT
Interval	IVL
Organization Name	ON
Person Name	PN
Physical Quantity	PQ
Ratio	RTO
Character String with Code	SC
Set	SET
Character String	ST
Telecommunication Address	TEL
Trivial Name	TN
Point in Time	TS
Address Part	ADXP
Entity Name Part	ENXP

In addition, following data types are not categorized as core data types from gap analysis. We will, however, support them from implementation perspective, but will not include them in our supported testing level, as compared to the list of data types above.

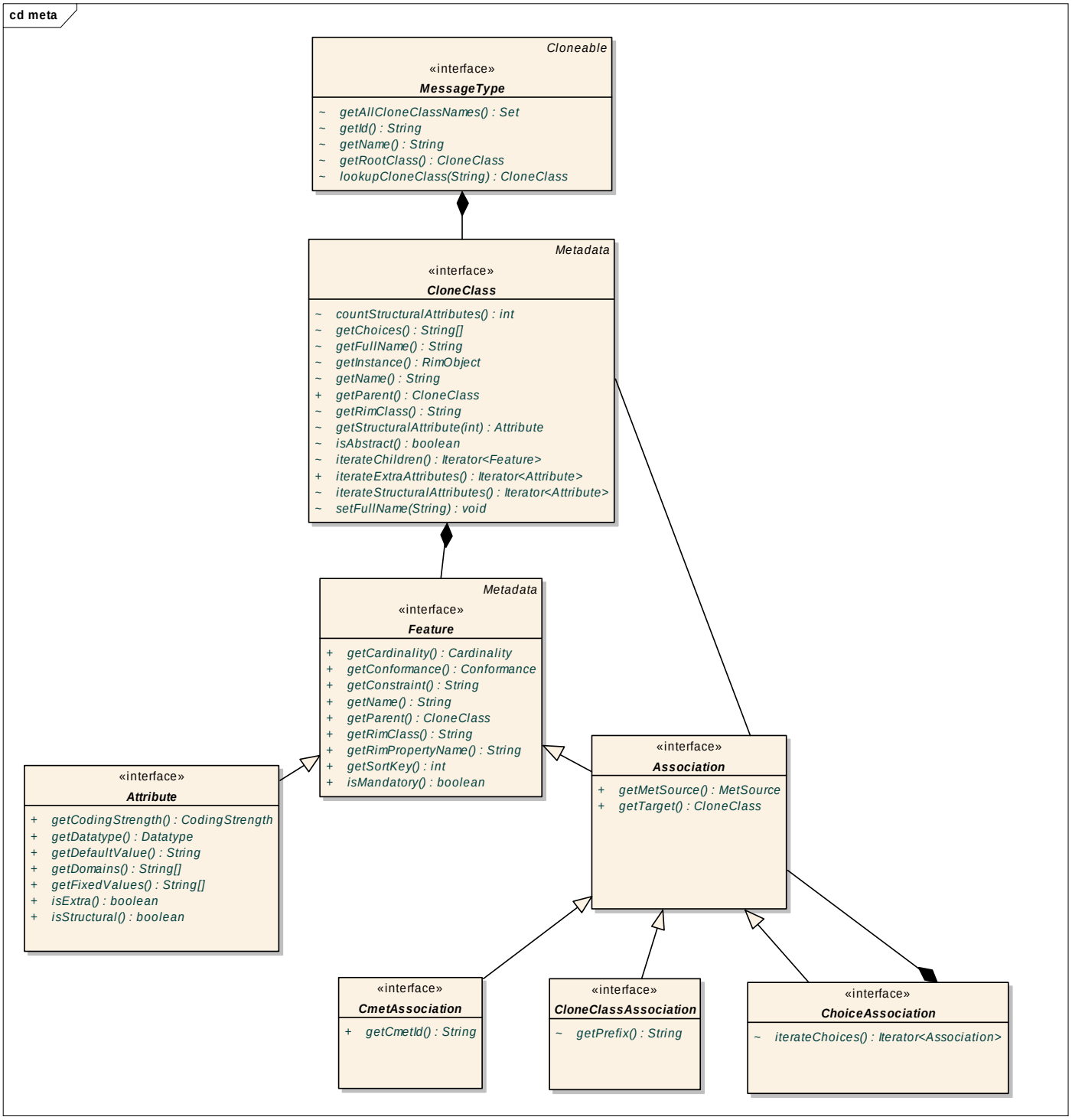
Name of Type	Type Code
Concept Role	CR
Event-Related Periodic Interval of Time	EIVL
Monetary Amount	MO
ISO Object Identifier	OID
Periodic Interval of Time	PIVL
Physical Quantity Representation	PQR
Unique Identifier String	UID
Universal Resource Locator	URL

For future reference, following list of data types have not been fully implemented in javasig code base as of 2005 Normative edition and none of them have been referenced on payload classes by RIM 2.07. Therefore, for this release, we will simply exclude consideration of them and focus on those listed above as our main task load.

Name of Type	Type Code
BooleanNonNull	BN
Calendar	CAL
Calendar Cycle	CLCY
Coded Ordinal	CO
GeneratedSequence	GLIST
History	HIST
History Item	HXIT
Non-Parametric Probability Distribution	NPPD
Parametric Probability Distributions over Physical Quantities	PPD<PQ>
Parametric Probability Distributions over Real Numbers	PPD<REAL>
Parametric Probability Distributions over Time Points	PPD<TS>
HL7 Reserved Identifier	RUID
Sampled Sequence	SLIST
DCE Universal Unique Identifier	UUID
Uncertain Value - Probabilistic	UVP

9.3 Meta Data Model

9.3.1 HL7 v3 Meta Data Model

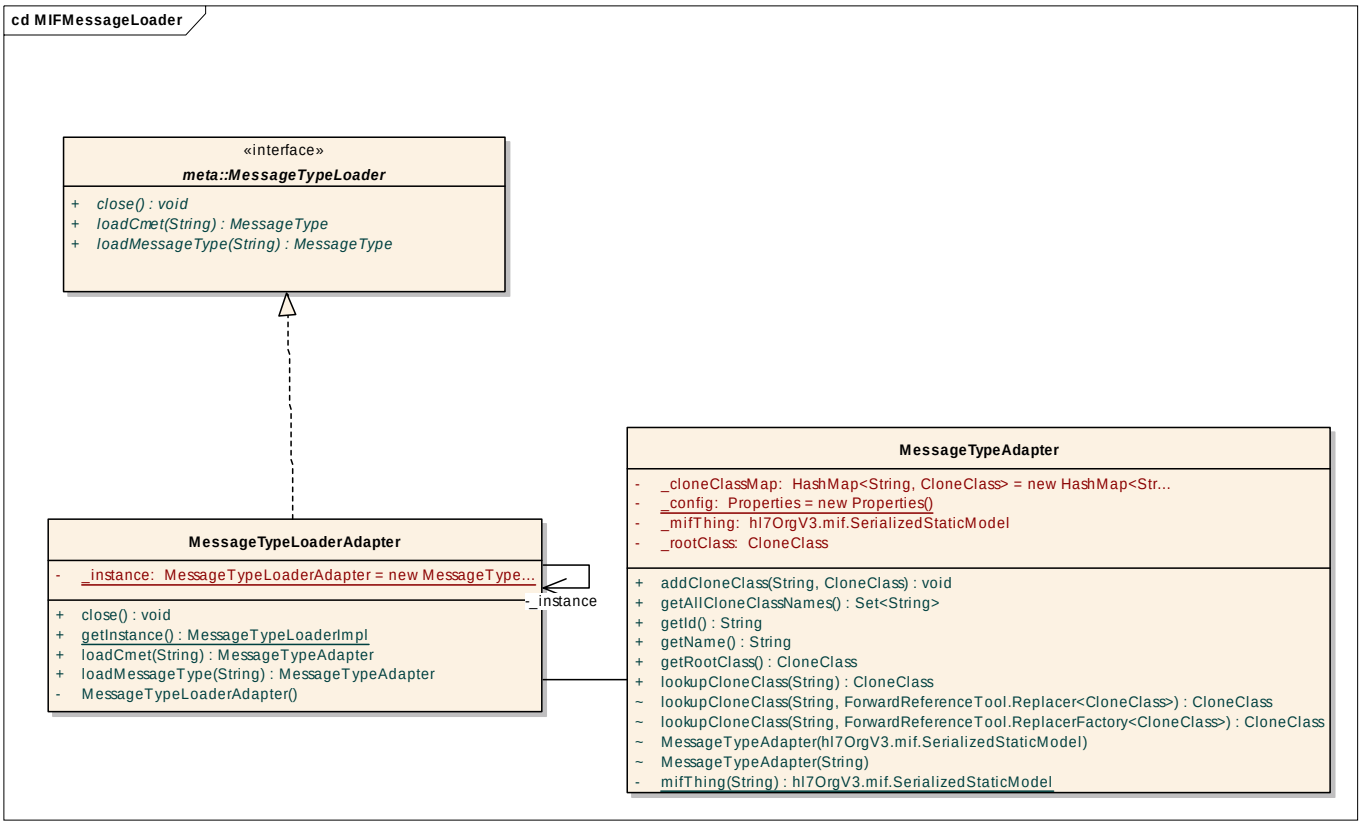


9.3.2 MIF File Loader

HL7 provides two formats Hierarchical Message Definition (HMD) and Model Interchange Format (MIF) for specifying message Meta data (structure, format, and constraints). They are both XML based. HMD is currenty depreciate by HL7. Start caAdapter version 1.3, the system only supports MIF.

The Meta Loader processes the Meta file via a JDOM parser. As key Meta elements are encountered within the input Meta file, Meta classes are created that mirror the attributes and other Meta class associations specified in the file. These Meta classes drive how the RIM object graph is built when messages are actually parsed.

Class MessageTypeLoaderAdpater is MIF file loader, which loads MIF file and converts it into Meta Object Graph.



9.4 Message Parser

The main parsing classes (content handlers) are found in the **org.hl7.xml.parser** package. Presenter classes are located one level higher.

The message parser utilizes a SAX parser when parsing the XM nodes of the input HL7 V3 message. As SAX events fire, data from the SAX event is used to query Meta classes for Meta data. The specific Meta data allows the parser to instantiate a node as a RIM object or HL7 data type object. The Meta data also denotes where to attach the instantiated object. Reflection is used to find the correct “setter” method to “add” the object to the growing RIM graph in memory. When an instance of a RIM class or data type is needed, it is created by a properties based factory.

Since there are many flavors of objects (RIM classes, data types, CMETS or Common Message Element Types) we need specialized content handlers. The specialized content handlers can be dynamically

"switched out" during message parsing. See the DynamicContentHandler class.

Specialized handlers (Presenters) are included for each RIM and data type class. Presenters are called by intermediate content handlers. Examples of these intermediate content handlers include: TreeContentHandler, DataTypeContentHandler, and SimpleTypeContentHandler, and others as well.

As Sax events fire, they are handled by MessageElementContentHandler which in turn can dynamically “suspendWith” to temporarily trade places with a specialized “presenter” class. The final object is handed back down the call stack to a “resultReceiver” with a call to “notifyResult”. Eventually, the returned partial result is added to the growing RIM graph by using reflection to find the correct “setter” method on the parent object.

Once the entire message is parsed, the resulting object graph is the entire in memory representation of the RIM Graph.

9.5 Message Builder

The message building classes are mostly found in the **org.hl7.xml.builder** package. The message builder’s design resembles the parser in that it relies on small specialized content handlers (which in this case are called “builders” instead of “presenters”).

If you look inside a presenter class you will find an inner class of type “builder”. This design choice was made so that once a presenter is written it is able to handle both message parsing and message building.

The message builder uses a clever “trick” to generate the XML output. Instead of writing many print statements, another approach was taken utilizing the "Identify transform" feature of XSLT transforms.

The “Identity transform” is a special case where the incoming XML message is sent through the XSLT transform machinery, but no actual transform is done. The newly generated output is exactly the same as the XML input: but the output isn't simply copied. The input generated SAX events and these events were used to create the XML output. Since no transform is performed, the resulting message is the same as the input message.

If you could read or parse the in memory RIM graph and pretend to be a SAX XMLReader, essentially generating the same events as if you were reading an XML stream, then you could use that reader in an identity transform to create XML output. That is the approach that was taken in the message builder (credit: Gunther Schadow).

The mechanism that masquerades as a SAX XML reader is made up of the classes XMLSpeaker and XMLRimGraphSpeaker. As the SAX events are emitted by “parsing” the in memory RIM graph, they are handled by “MessageBuilders”. The message builders are then dynamically switched for one another as appropriate. Finally one of the builders emits a SAX event to an internal member of type “org.xml.sax.ContentHandler”. This member happens to be the "_contentHandler" member from the abstract class XMLSpeaker. Once this event fires, the XSLT transform takes over and the XML for that particular node is created.

10. Human Computer Interface

Topics in this chapter include:

10.1 Application Storyboards and Descriptions

The caAdapter Mapping Tool uses a document-oriented paradigm where up to four files of different types can be open at the same time, each within its own tab in a single window. This keeps memory and window management to a minimum, which is a benefit to both developers and users. There are four different types of "documents" or tabs depicted with mockups which are described in the sections indicated:

1. CSV specification tab (see section *10.1.1 CSV* Specification on page 48)
2. HL7 v3 target specification tab (see section *10.1.2 Target* Specification on page 52)
3. Mapping specification tab (see section *10.1.3 Mapping* Specification on page 55)
4. HL7 v3 XML instances tab (see section *10.1.4 CSV-to-HL7 Conversion* on page 58)

One tab for each type of tab may be open at any given point. The tab name displayed is the name of the main file generated in the tab (for example, `090102.map`) or it is labeled `untitled.<ext>` where `<ext>` is the appropriate file extension for that type of tab (see section *11.5 File Extensions* on page 69 for more on file extensions).

The mockups contained in this section are provided as a **suggested screen design**. They are illustrated at a high level, and they do not prescribe all detailed aspects of the GUI design or cover all functional requirements for this system. Since this is a living document, the mockups in this document and the actual GUI may change over the life of the development phase as new considerations come to light or better mechanisms are identified.

Generally speaking, buttons apply to the subject matter in the panel in which the button is located. The menu across the top of the window can also be used to accomplish some of the same tasks as buttons. Also, the functions that will be enabled in a later phase of development have been grayed out but included to show the direction the tool will take in later development phases. Developers will implement common features such as scrollbars, where needed, and in many cases these have been left out of the mockups to keep the diagrams simple.

The following sections illustrate one or more mockups for each of the types of tabs, along with a description of the features or functions contained therein.

10.1.1 CSV Specification

The purpose of the CSV specification tab is to allow the user to identify the hierarchy of segments and fields that describe an incoming CSV file containing data that must be converted to HL7 v3 XML messages. The hierarchy is visually represented using the common GUI feature of an expandable/collapsible tree structure.

The main design aspect of the CSV specification tab is to separate the tree structure in the left-hand panel from the details and functions in the right-hand panel. The tree structure displays the hierarchy of segments and fields that represent the way data in the source CSV files are organized. Typical features of the tree structure are used, such as dragging and dropping an element to another location in the tree, or the ability to expand and collapse a branch of the tree using the "+" and "–" symbols respectively. Sections in the right-hand panel allow the user to work with the specification on the left. The following sections describe how these features work and how this tab is accessed.

10.1.1.1 Basic Features of the CSV Specification Tab

The first tab mockup (see *Figure 10-1*) shows the basic functions performed in the CSV specification process.

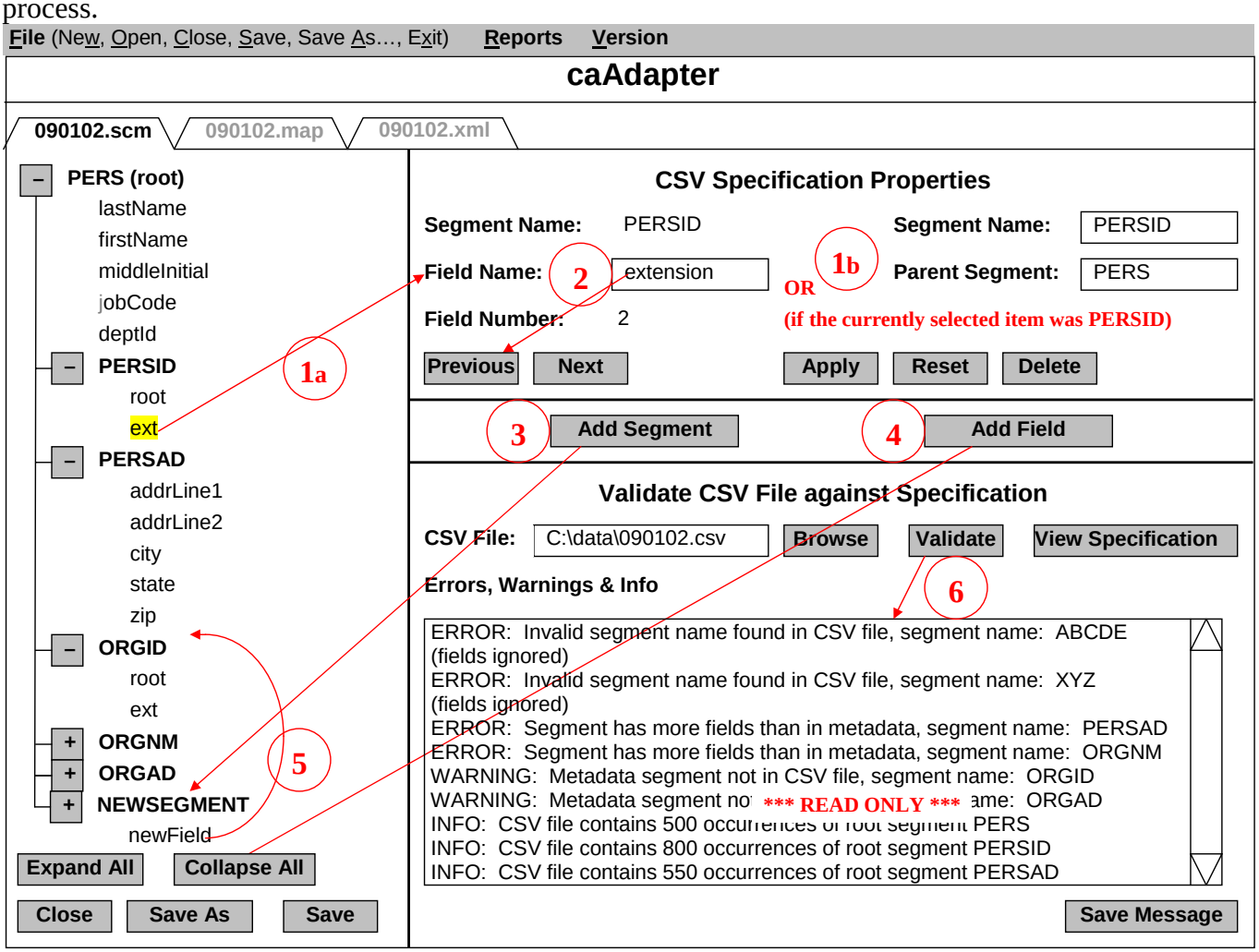


Figure 10-1 Basic features of the CSV Specification Tab screen mockup

The basic functions indicated in *Figure 10-1* are as follows:

- When the user clicks on an element in the tree structure, the details of that element appear in the Properties section of the right-hand panel. The type of item that is clicked drives what details get displayed: (1a) shows the details for a field, while (1b) shows the details for a segment.
- The ability to edit the name of a field or segment is indicated by (2), where after making the changes, the user clicks on the **Apply** button to make the change display in the tree.
- The ability to add a new segment (3) or field (4) is provided by selecting **Add Segment** or **Add Field**, respectively, in the right-hand panel. New segments have a default name of **NEWSEGMENT**, **NEWSEGMENT2**, **NEWSEGMENT3**, etc. and users can edit these names as desired, but the system always saves them in uppercase. This is to ensure that they are always unique regardless of case. New fields have a default name of **newField**, **newField2**, **newField3**, etc. and users can edit these items as well. The preferred convention for field naming is camelCase.
- Dragging and dropping a field or segment to another area in the tree allows the user to rearrange the tree contents as indicated by (5). Moving a segment takes its complete sub-tree with it. Users may not drag-n-drop the root segment; it must remain as the root, but its fields may be moved.
- Once the user is satisfied with the specification, they can test a CSV file against it to see how well they correspond to one another. The user selects the file using the **Browse** button and then clicks the **Validate** button at (6) to generate messages about the correspondence between the CSV file and the specification. This process automatically saves the specification to a temporary location for the process of validation. The user may also save the results by clicking **Save Message** to address any needed changes.

10.1.1.2 Creating and Saving a CSV Specification

The next mockup (see *Figure 10-2*) shows how the user can create and save a new CSV specification.

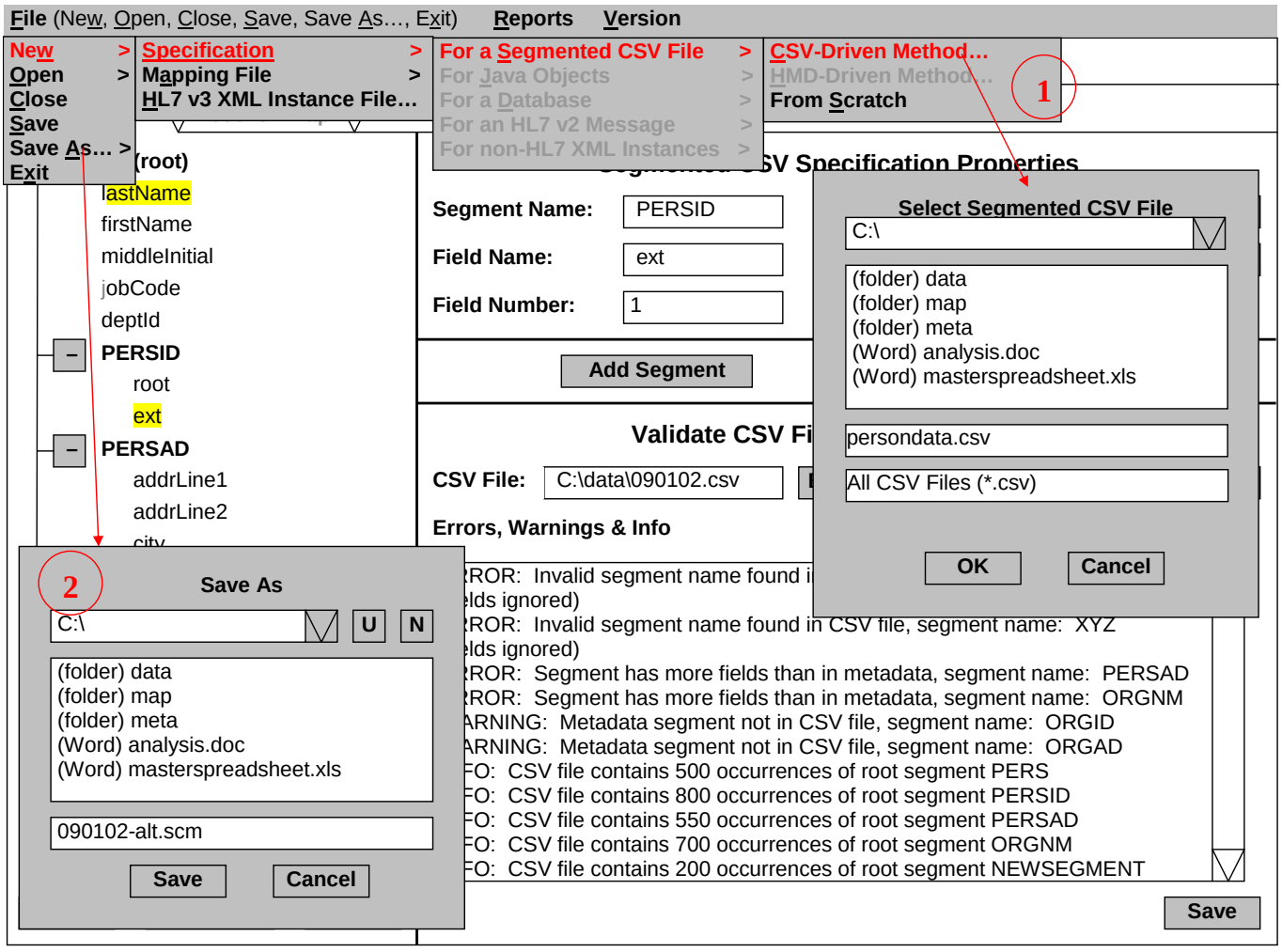


Figure 10-2 Creating and saving a CSV Specification screen mockup

The basic functions indicated in *Figure 10-2* are as follows:

- The **File > New** option has a series of sub-menus that direct the user to the type of file they want to create and the method to use to create it. In *Figure 10-2*, the user has chosen the option to create a new specification file based on an existing CSV file as indicated by (1). After the user selects the CSV file, the system parses the file, identifying all the segments and counting the maximum number of fields per distinct segment name. The newly skeletonized tree is displayed in a new specification tab in the left-hand panel using the identified segment names and the default field names (described in section *10.1.1.1*).
- The other option for creating a new specification specification is indicated in the diagram (but not shown separately) by the **From Scratch** option in the last sub-menu. This option opens the CSV specification window with an empty tree except for an initial root segment which is **NEWSEGMENT** by default. The user edits the name and adds dependent fields and segments using the buttons provided.
- When a user is finished working on a specification file, the **File > Save** or **File > Save As** feature, indicated by (2), is used to create an XML-like file describing the structure of the tree structure. The file is portable and can be opened by the same or another user later.

10.1.1.3 Opening an Existing CSV Specification

The user can open a CSV specification that was saved earlier by choosing the **File > Open** option (see *Figure 10-3*). This option has sub-menus that direct the user to choose the type of file to open. When the user selects the specification file to open (as indicated in *Figure 10-3*) and clicks the **OK** button, a new tab is opened with the specification displayed in the tree. The user continues making changes as described in section *10.1.1.1* and section *10.1.1.2*.

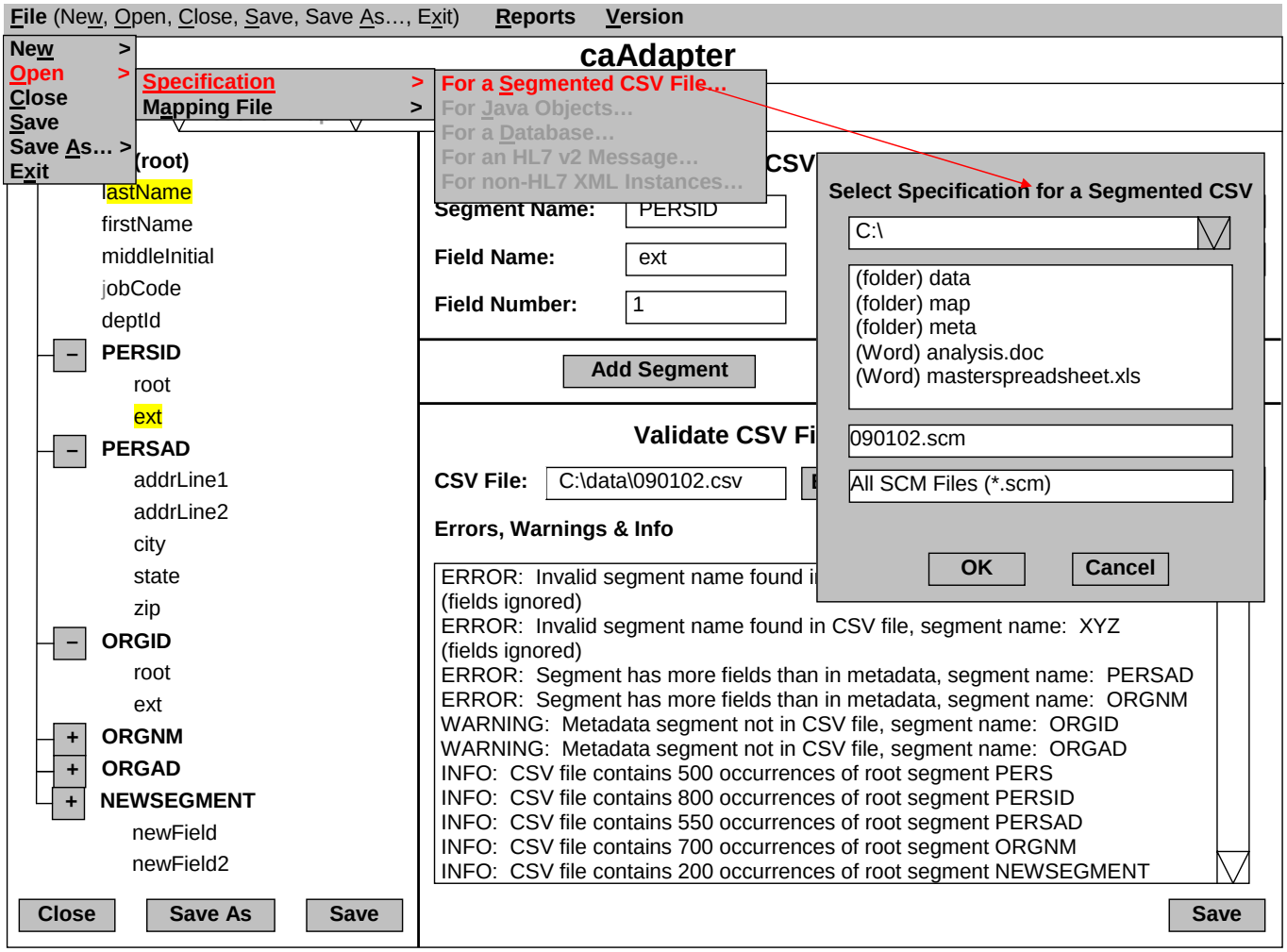


Figure 10-3 Opening an existing CSV Specification screen mockup

10.1.2 Target Specification

The purpose of the target specification tab is to allow the user to customize the hierarchy of the HMD clones and attributes to fit the needs of an incoming CSV file. The hierarchy is visually represented using the common GUI feature of an expandable/collapsible tree structure.

The main design aspect of the target specification tab is to separate the tree structure in the left-hand panel from the properties in the right-hand panel. The tree structure displays the hierarchy of clones, attributes and fields that represent the HMD tailored to a particular source specification. Some typical features of the tree structure, such as dragging and dropping an element to another location in the tree, are NOT used

in the target specification screen because HL7 prescribes the basic HMD structure and there are only a few ways it can be customized for mapping purposes: element multiplicity, abstract data type specification, choice boxes, and user-defined default values. The following is a description of how these features work and how this tab is accessed.

10.1.2.1 Basic Features of the Target Specification Tab

The Target Specification tab is shown in *Figure 10-4*. This tab is labeled by the name of the specification file. Similar to the Source Specification tab, this tab is based on a tree structure on the left and a panel containing element properties on the right.

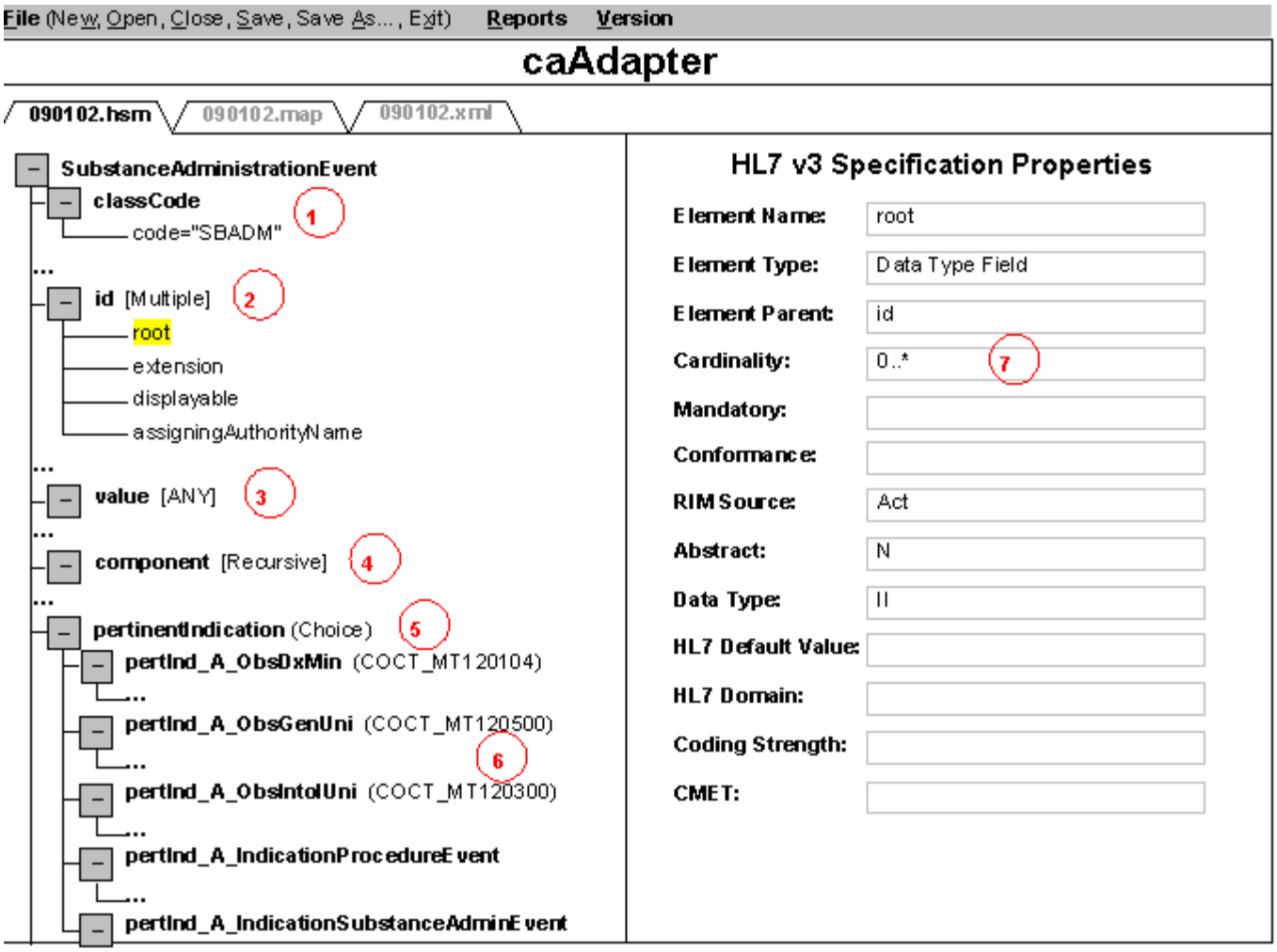


Figure 10-4 Target specification tab screen mockup

The mockup in *Figure 10-4* shows the basic conventions for conveying information pulled from the underlying HMD as described below.

- The default values for all structural attributes are displayed as an = assignment to the right of the applicable data type field (1). These values cannot be changed by the user and are required as part of the HL7 message specification.

- Some message elements with have either a cardinality of 0..* or 1..* and/or has a data type that involves a collection (for example, SET, BAG, LIST). These items appear in the HL7 v3 XML as simple repeats of the element. To accommodate the possible requirement of mapping more than one source element to the same target element, the tab allows the user to add multiples of such elements which are indicated with a **[Multiple]** label (2) following the element name. Right-clicking on this element name allows the user to add a multiple.
- HL7 message developers occasionally do not specify a particular data type to use when populating some attributes, leaving it up to the source system to govern what data type to use by assigning a data type of ANY or QTY. These are abstract data types and the user is required to assign a specialized data type to such attributes. When an ANY or QTY is found in an HMD, the target specification tab shows the element name and a label of **[ANY]** (3) or **[QTY]** to the right. Right-clicking on this element will allow the user to select a particular data type.
- Large messages or Common Message Element Templates (CMETs) often have recursive associations on them. Such relationships cannot be expanded ad infinitum in the target specification tab, so the convention for indicating the possibility of a recursive relationship is to add the label **[Recursive]** (4) beside the element name. Right-clicking on this element name allows the user to expand one recursive "child" generation at a time.
- Choice boxes in HL7 messages present a challenge to mapping. The limitation for the current choice implementation is the ability to choose a single option to which all logical records in the source file may be mapped. This decision is made in the mapping tab rather than in the target specification tab, but the indication of the presence of a choice is indicated in the here with a **(Choice)** label (5). All options are displayed in the target specification tab.
- HL7 analysts become familiar with the reusable components, or CMETS, that make up messages, not just the messages themselves. The ability to indicate that an element represents the beginning of such an included CMET may be very useful to analysts. This tab indicates such elements by including a label to the right of the element name where the CMET code (for example, **COCT_MT120104**) is enclosed in parentheses (6).
- When a user clicks on the name of an element, the properties of that element are displayed in the right-hand panel, **Target Specification Properties** (7). The properties include fields that may or may not apply to the level or type of the element displayed. For instance, a data type field such as an **id.root**, would not have a **CMET** name, or would not be **Mandatory** (since data type fields are theoretically all optional). For attributes that apply to an element's type, the associated data is displayed. This data also helps the system determine when an element may have multiples (based on data type and cardinality). These properties should reflect the values found in the HMD. Only property fields when the element has an abstract data type of **ANY** or **QTY** are editable. The data type field, **Element Type**, is editable via a dropdown or popup window (exact implementation details are up to the developers).
- Saving a target specification file is accomplished by using the **File > Save** or **File > Save As** option (similar to saving a Specification file as shown in *Figure 10-2*).
- Creating a new target specification file is accomplished by using the **File > New**> **Target Specification File** option (similar to creating a Specification file as shown in *Figure 10-2*). The sub-menu allows the user to select a message type from a dropdown list of HL7 messages that have been validated with the caAdapter Mapping Utility by the caAdpater development team.

10.1.3 Mapping Specification

The purpose of the mapping specification tab is to allow the user to assign fields in a CSV specification to fields in an HL7 v3 XML message. For both the source side (the CSV specification) and the target side (the expanded HMD), the hierarchy is visually represented using an expandable/collapsible tree structure.

The main design aspect of the mapping specification screen is two tree structures with the source specification in the left-hand panel and the expanded HMD in the right-hand panel. The tree structure for the specification is identical to that displayed in the CSV specification screen as shown in *Figure 10-3*. The tree structure for the expanded HMD is likely to be deeper, displaying the elements, sub-elements, etc., down to the data type field level. In addition to the two tree panels, the center panel shows the lines that indicate the mapping between source fields and target fields and any functions that are used in the mappings. The following sections describe how these features work and how this screen is accessed.

10.1.3.1 Basic Features of the Mapping Specification Tab

The first mapping specification mockup (see Figure 10-5) displays the basic features of the tab: the two tree panels, center mapping panel, functions panel and properties panel. Each of the four panels can be resized by selecting the edge of the panel and dragging. The tree structures are essentially read-only apart from the mappings and the default values that may be defined for target fields.

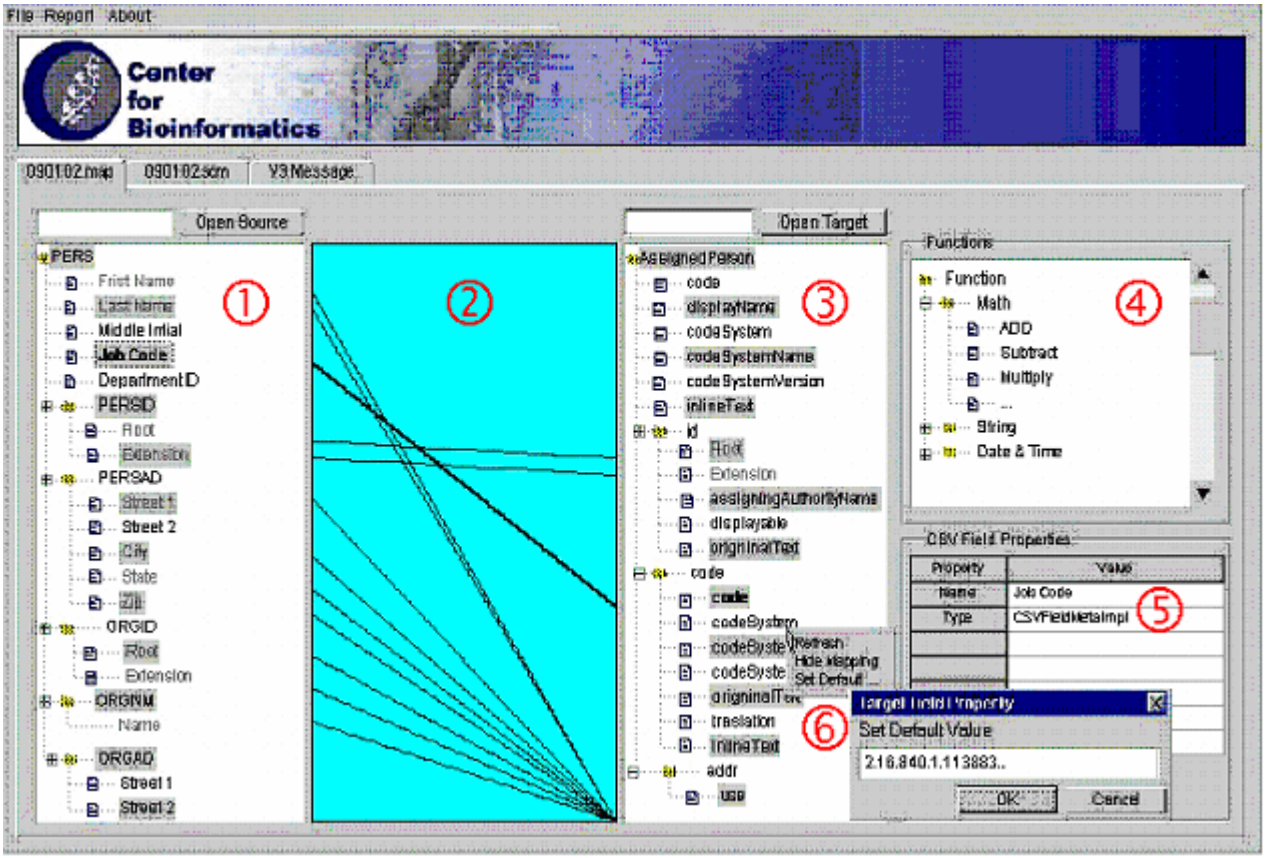


Figure 10-5 Basic features of the Mapping Specification Tab screen mockup

The basic functions shown in Figure 10-5 are as follows:

- Mappings are created by dragging and dropping source fields (1) onto target fields (3) that results in mapping lines between the source and target fields (2). When a field is mapped, it is visually changed, such as being grayed out. The lines between source and target sides can be turned off if they become too numerous to be readable.
- A tree of available functions and a table of properties for the selected element can be viewed in the **Functions** (4) and **Properties** pane (5).
- Default values can be defined for target fields regardless of whether there is or will be a mapping for that field (6). To set a default value, the user selects the field, right clicks to bring up the context menu, and selects **Set Default** to display the Target Field Property dialog. The user enters the value in the **Set Default Property** field, clicks **OK** and the value is displayed in the Properties pane. If the field is mapped, then the default value is only used if the value coming from the corresponding source CSV field is null. If the field is not mapped then the default value is used for each occurrence of its element.

10.1.3.2 Using Functions in Mappings and Multi-Occurring Target Elements

Sometimes an incoming value needs manipulation before being used in the target message as shown in Figure 10-6.

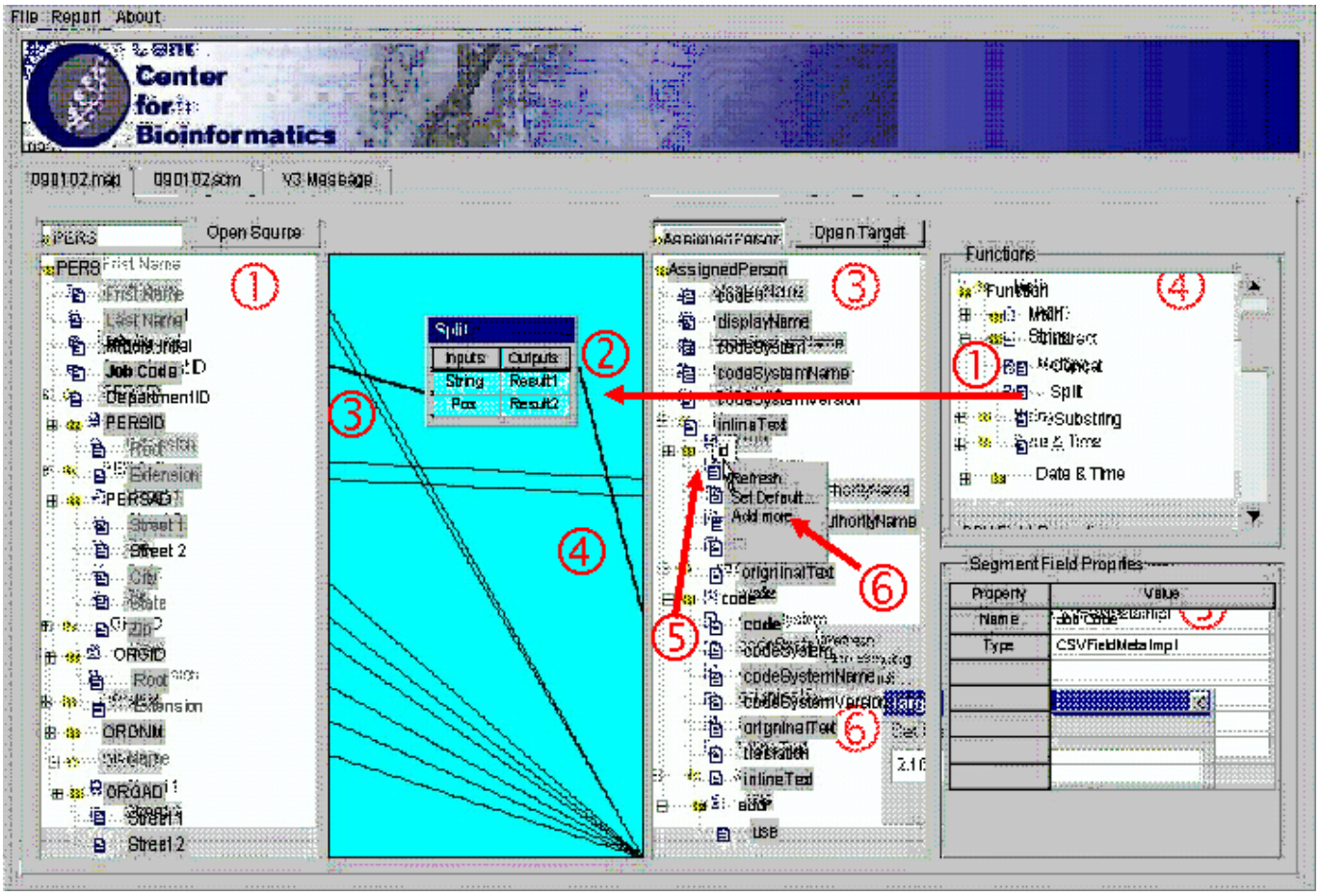


Figure 10-6 Using a function in a mapping and multi-occurring target elements screen mockup

When it is not possible to do such manipulations prior to creating the CSV file, the user may use a function in the mapping to effect a change (see Figure 10-6).

- Functions can be added by dragging and dropping a function from the **Functions** panel (1) to the mapping panel. This results in the function box being added in the mapping panel (2).
- This function box can be moved around the mapping pane as convenient for attaching the mapping lines. Mapping lines are attached by dragging and dropping the source fields onto the input parameters (3), and the output parameters onto the target fields (4). The mapping lines go from the source fields into the function box and out of the function box to the target fields.

Some elements in the target tree may be items that can occur more than once in their parent element. This is particularly important for mappings that originate in several different segments or fields in the source side that map to the same fields in the target side. An example of this is address parts: some source systems provide address data in distinct fields that hold a specific address part (for example, city, state, zip), while HL7 models address parts as the AddressPart (ADXP) data type that occurs in a list in the PostalAddress (AD) data type. The target side will then need multiple occurrences of the ADXP fields to have a different default value assigned to the part type for each type of address parts in the source side. To accommodate this, the design proposes the following:

- Highlight the element that needs to be repeated (5) and select **Add More** from the context menu for that element (6).
- This adds another ADXP element to the target tree structure giving the user the opportunity to define a different default value for partType and mapping a different source field to the inline Text field. This convention should work for other situations where multiple occurrences of an element in the target tree are required to properly map source data.

10.1.3.3 Creating, Opening and Saving Mapping Files

Due to the similarities between creating, opening and saving specification files and mapping files, tab mockups of these processes have not been included for mapping files but references are made to the appropriate figures for the specification files.

Creating a new mapping file is accomplished by using the **File > New** option (similar to creating a Specification file as shown in *Figure 10-2*). The sub-menu allows the user to select a new mapping file and then choose the mapping type. Currently, only the **CSV to HL7** option is implemented, so the other options are grayed out. This action opens a new mapping tab and displays the mapping tab with empty source and target panels until the user selects the source specification file and the target HMD file from the file system. Once these are chosen, their respective panels are populated with the tree structures. The user can then start the drag-and-drop mapping processes.

Opening an existing mapping file is accomplished by using the **File > Open** option (similar to opening a Specification file as shown in *Figure 10-3*). Once the file is selected, the source and target trees are displayed along with any existing mappings. The user can then continue to work on the mappings.

Saving a mapping file is accomplished by using the **File > Save** or **File > Save As** option (similar to saving a Specification file as shown in *Figure 10-2*). The **File > Save** or **File > Save As** allows the user to specify the file name and target directory.

10.1.4 CSV-to-HL7 Conversion

The purpose of the HL7 v3 XML instances tab is to allow the user to view both the XML instances converted from a CSV file and the error, warning and informational messages that are generated during the conversion process. Each CSV file may have one or more logical records which results in a corresponding number of XML instances (or more depending on the structure of the mapping). Therefore, the output needs to be navigable between the instances.

The main design aspect of the HL7 v3 XML instances screen is two scrollable text fields where instances and messages are displayed. The following sections describe how these features work and how this screen is accessed.

10.1.4.1 Starting the Conversion Process

To start the conversion process of a CSV file into HL7 v3 XML instances, the **File > New** option is used (see *Figure 10-7*).

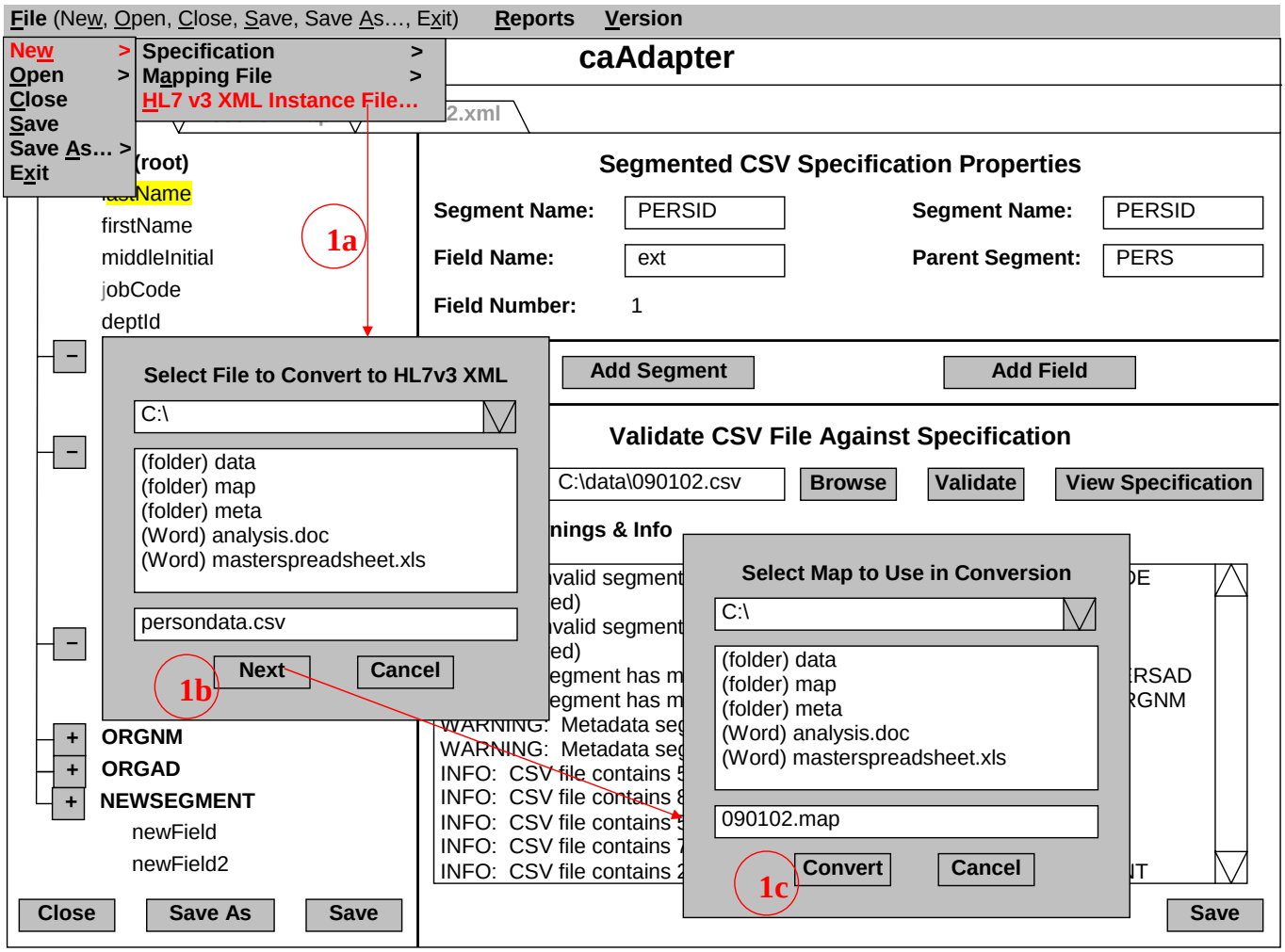


Figure 10-7 Starting the conversion process screen mockup

To convert a CSV file into HL7 v3 XML instances, the following steps are done (see *Figure 10-7*).

- The user selects the **File > New** option, and then selects **HL7 v3 XML Instance File** (1a) from the sub-menu.
- This creates a pop-up window where the user is prompted to select the CSV file to be converted (1b).
- The user is then prompted to select the mapping file to use in the conversion process (1c).
- Once the user clicks the **Convert** button (1c), the system generates or attempts to generate HL7 v3 XML instances from the CSV file using the `.map` file selected.

Note: There is no **File > Open** option that corresponds to XML instances since the user will want to generate fresh messages based on the current selection of CSV and mapping files.

10.1.4.2 The HL7 v3 XML Instances Tab

The two main features of the HL7 v3 XML instances tab (see *Figure 10-8*) are the two scrollable text fields containing an XML instance and the associated error, warning and/or informational messages generated during the conversion process.

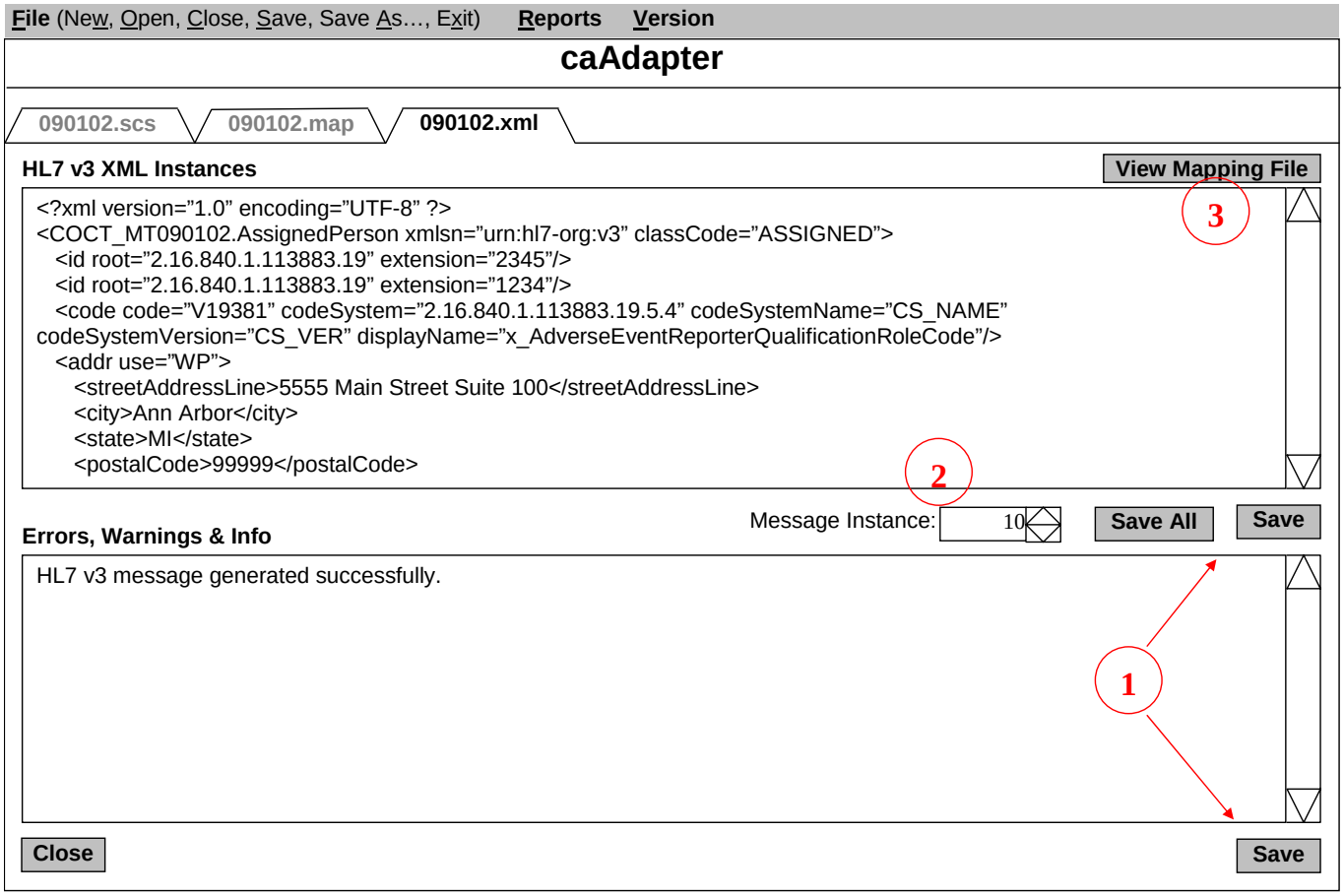


Figure 10-8 HL7 v3 XML instances tab screen mockup

Following are the main features of the HL7 v3 XML instances tab (see *Figure 10-8*):

- Both text fields can be saved individually (1) for future reference by clicking **Save**, which may be especially useful during the process of fine-tuning a map.

- The user can cycle through the XML instances one at a time using the **Message Instance** counter (2) located below the XML message text box. Alternatively, the user can enter the number of a particular message of interest. As the XML instances change, the error messages change so that they are always in synch with one another.
- The user can view the contents of the `.map` file used, by clicking the **View Mapping File** button (3) which displays a separate window where the user can view the XML-like `.map` file.

10.2 Matrix of Functions and Input/Output

Listed below are the functions used by the mapping tool in the following tables:

- *Table 10-1 Math functions* on page 60
- *Table 10-2 String functions* on page 61
- *Table 10-3 Date functions* on page 62

Math Function	Input A	Output	Example
Add: Sum of two numbers	Value (long) Value (long)	SUM (long)	Value = 40 Value = 80 SUM= 120
Subtract: The difference between two numbers.	Value1 (long) Value2 (long)	Difference(long)	Value1 = 50 Value2 = 10 Difference= 40
Multiply: Product of two numbers	Value (long) Value (long)	Product (long)	Value = 10 Value = 6 Product = 60
Divide: Division of two numbers	Dividend (long) Divisor (long)	Quotient (long)	Dividend = 50 Divisor = 10 Quotient = 5
Round: Rounds input to nearest integer	Value(long)	Result(long)	Value = 8.43 Result = 8

Table 10-1 Math functions

String Function	Input A	Output	Example
Concatenate: Linking together or joining two strings	String1(String) String2(String)	Result(String)	String1=NCI String2=CB Result= NCICB
Split: Split a string at a character position. Inputs are the string and position.	String (String) Pos(int)	Result1(String) Result2(String)	String1=NCICB Pos=4 Result1= NCI Result1= CB

String Function	Input A	Output	Example
Substring: Extract a substring based on character position. Inputs are the string, start position, and end position	String(String) StartPos(int) EndPos(int)	Result(String)	String=NCICB StartPos=1 EndPos= 3 Result= NCI
Instring: Find the position of a substring within a string. The inputs are the string, pattern. The output is the position of the beginning of pattern within the string	String(String) Pattern(String)	Result(int)	String=NCICB Pattern=CB Result= 4
Length: Length of a sting in characters.	String(String)	Length(int)	String=NCICB Length= 5
Replace: Find and replace a substring within a string. This will replace all substrings.	String(String) Find(String) Replace(String)	Result(String)	String=NCICB Find=CICB Replace=IH Result= NIH
Upper: Format a string to uppercase	String(String)	Result(String)	String=Ncicb Result=NCICB
Lower: Format a string to lowercase	String(String)	Result(String)	String=NCIcb Result=ncicb
Initcap: Format the first character of a string to uppercase. Each character following a <Space> within the string will formatted uppercase	String(String)	Result(String)	String=ncicb Result=Ncicb

Table 10-2 String functions

Date Function	Input A	Output	Example
Format Date: (input = date in, from format, to format; output = date out) **Note:** this requires a standard set of date format elements (e.g. like Oracle date formats)	Date(int) FormatIn(String) FormatOut(String)	Result(int)	Date=07301991 FormatIn=MMDDYYYY FormatOut=YYYYMMDD Result= 19910730

Date Function	Input A	Output	Example
Time Between: Returns the number of same date increment between two dates based on a Gregorian calendar. The inputs are date 1, date 2, format in and unit out. The outputs are time quantity and time unit.	StartDate(int) EndDate(int) Format(String) Increment(String)	Result(int) Increment(string)	StartDate =19910730 EndDate =20050715 Format =YYYYMMDD Increment =Days Result1 = 5384 Result2 = Days

Table 10-3 Date functions

10.3 Report Design

The caAdapter application is requirered to provide printable reports. These reports assist analysts in creating source to target specification by providing attribute information. For this phase, the mapping tool will support the source report and the mapping report. Future phases will include other reports.

10.3.1 Source Report

The source report displays attributes of the CSV specification file. These attributes include field number and structure of the segments.

SOURCEOBS	codeSyster	code	text	time	value
OBSID	root	extension			

Table 10-1 Source Report Example

10.3.1.1 Segment Structure

Segment parent/child relationships are shown in the report as indents. Child segments names are indented 3 spaces from their parent segment. In the above example, the OBSID segment is a child of the SOURCEOBS segment. Sibling segments are indented at the same level.

10.3.2 Mapping Report

The mapping report displays the status of mapping from source specification to target specification. The report will include five categories, namely, mapped source to function inputs, mapped source to target, mapped function output to target, unmapped source, and unmapped target.

11. File Formats

This chapter contains the format for the various files used by the mapping tool.

Topics in this chapter include:

11.1 CSV Specification

11.1.1 Overview

CSV specification describes the structure of a CSV instance. In essence, it is a CSV specification of a CSV instance in the same way an XSD is a specification of an XML instance. The CSV specification is based on common concepts found in EDI, CSV and HL7 v2.x-related files. It is an assumption in the requirements document for this phase of development that:

"Source data systems provide data in CSV flat file formats with the following characteristics:

- File contents are organized into multi-line logical records.
- Each line, called a *segment*, begins with an identifier, called a *segment name*, and is terminated by a “new-line” character.
- Each segment has one or more data items, called *fields*, which follow the segment name and terminates by commas (except for the last field on the line that uses the segment terminator).
- Segments may occur more than once in the same logical record, except for the first or *root* segment, which always indicates the beginning of a new record.
- Segments are related to one another in a parent-child hierarchy that documents the one-to-many nature of the association between related data items.
- A CSV file may have one or more logical records. Each of these is terminated by the beginning of the next record (a new root segment) or the end of file.
- The intention is that each logical record will become one single HL7 v3 XML message instance."

To document this structure, the CSV specification uses an XML format that has three main elements:

1. `<csvMetadata>`
2. `<segment>`

3. <field>

There can only be one root , but within it there can be any number of dependent elements and any number of <field> elements. All <field> elements have a column number assigned which corresponds to the second, third, etc. column in the CSV file (the first is the segment name which is considered column 1). The field names are informational and are not used in the mapping file (see section *11.4 Mapping Specification*) where only the segment name and column number is referenced.

11.1.2 Specification File Example

Following is a specification file example.

```
<?xml version="1.0"?>
<csvMetadata version="1.2" uuid="1111ABCDEFGH">
  <segment name="SAE" uuid="1111ABCDEFGH">
    <field column="1" name="code" uuid="1111ABCDEFGH"/>
    <field column="2" name="effectiveTime" uuid="1111ABCDEFGH"/>
    <field column="3" uuid="1111ABCDEFGH"/>
    <field column="4" uuid="1111ABCDEFGH"/>
    <segment name="CONT" uuid="1111ABCDEFGH">
      <field column="1" name="drugCode" uuid="99999"/>
      <field column="2" name="drugFormCode" uuid="1111ABCDEFGH"/>
    </segment>
    <segment name="OBSDX" uuid="1111ABCDEFGH">
      <field column="1" name="pertInfo1PauseQty" uuid="1111ABCDEFGH"/>
      <field column="2" name="obsDxId" uuid="1111ABCDEFGH"/>
      <field column="3" uuid="1111ABCDEFGH"/>
     <segment name="DXAUTH" uuid="1111ABCDEFGH">
           <field column="1" name="assignedEntId" uuid="1111ABCDEFGH"/>
            <!-- a space is provided in this example so the schema
            generator would not put restrictions on this attribute.-->
           <field column="2" name="2nd assignedEntId" uuid="1111ABCDEFGH"/>
           <field column="3" uuid="1111ABCDEFGH"/>
       </segment>
     </segment>
  </segment>
</csvMetadata>
```

11.2 HL7 v3 Specification

11.2.1 Overview

The HL7 v3 specification used to define the HL7 v3 information is based largely on the HMD for the target HL7 message. It uses four main types of nested elements:

1. <hl7v3meta>
2. <clone>
3. <attribute>
4. <datatypeField>

11.2.2 HL7 v3 Specification File Example

Following is an HL7 v3 specification file example.

```
<?xml version="1.0" encoding="UTF-8"?>
<hl7v3meta xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" messageId="COCT_MT090102"
xsi:type="hl7v3meta">
   <clone clonename="AssignedPerson" uuid="9d49d1b7-47af-4d34-9428-33d548091cf0">
      <attribute name="type" uuid="81b40671-5749-4631-83fa-6df979c4d0ba">
         <datatypeField name="code" uuid="2a6447ff-16bd-4c2b-867a-54699e33e58e"/>
         <datatypeField name="displayName" uuid="f4ad189f-9af3-4ac7-8f0a-b344e629aa0d"/>
         <datatypeField name="codeSystem" uuid="fa150447-3d07-45f5-8415-981b5feb2b4a"/>
         <datatypeField name="codeSystemName" uuid="be00b2f3-36a1-44b4-9e12-03f9e9cf7e8c"/>
         <datatypeField name="codeSystemVersion" uuid="652efa49-76fc-421d-9413-cec98d775ed8"/>
         <datatypeField name="inlineText" uuid="141dad9e-c613-416a-85fd-884e7338d0ce"/>
      </attribute>
      <attribute name="classCode" uuid="c2d4fa6e-fd91-4d6d-9134-639f5736386f"/>
      <attribute name="id" uuid="23606fee-bdd5-4520-97b4-be124827e2d2">
         <datatypeField name="root" uuid="0dbe07b2-f792-459e-b24b-bd3dc3d7530b"/>
         <datatypeField name="extension" uuid="1be8fbed-688a-412c-8593-169f35d83d9f"/>
         <datatypeField name="assigningAuthorityName" uuid="58948600-850e-4383-96b0-
605ddb7b3adc"/>
         <datatypeField name="displayable" uuid="da5bfdb3-1949-47dd-b7e6-789732304bca"/>
         <datatypeField name="inlineText" uuid="8f30af82-d6a3-42ec-b961-bd7640261913"/>
      </attribute>
      <attribute name="code" uuid="a64d3c1a-902d-481c-9126-5c6256c6889f">
         <datatypeField name="code" uuid="268556b5-f850-48cb-8330-092c288c1b4b"/>
         <datatypeField name="codeSystem" uuid="46704dfc-a448-4030-86c9-9161c3308bc9"/>
         <datatypeField name="codeSystemName" uuid="e5d906c9-234d-43e2-8437-ecf8045138ff"/>
         <datatypeField name="codeSystemVersion" uuid="cdbb1d64-9ffb-42aa-8345-5e1e041b5fe0"/>
         <datatypeField name="displayName" uuid="9b6ce6e2-7d86-4fa6-a295-e8c13b775f71"/>
         <datatypeField name="originalText" uuid="0039798c-6660-4d6a-a508-519a2ce42e13"/>
         <datatypeField name="translation" uuid="b5685867-841d-48af-92b5-fd160dd95987"/>
         <datatypeField name="inlineText" uuid="8e8ff2e9-e838-40a3-86ac-e50264956bf5"/>
      </attribute>
      <attribute name="addr" uuid="33cd65d0-b512-429f-bfb2-b1e76a332d3f">
         <datatypeField name="use" uuid="11faa020-1165-4ebd-a123-612c1edb6a58"/>
         <datatypeField name="validTime" uuid="a9a5d9d0-4804-4277-a683-c4b241b48816"/>
         <datatypeField name="isNotOrdered" uuid="14a0b627-f1a4-497f-b636-df482de9f377"/>
         <datatypeField name="inlineText" uuid="48713ba2-b2ab-44bc-8a4a-b49ce5049215"/>
         <attribute name="delimiter" uuid="21be5f92-b321-4f77-a11b-18fae58c54c8">
            <datatypeField name="inlineText" uuid="1731d72c-747b-447c-a2b2-db0ed31bd517"/>
         </attribute>
         <attribute name="county" uuid="14b4b15c-081b-4da1-8671-b851703837d2">
            <datatypeField name="inlineText" uuid="6f8455da-cc47-4677-8af6-da8f67ca130d"/>
         </attribute>
         <attribute name="postalCode" uuid="9fa1242f-73f1-400b-b2ca-aee3b4bb16b9">
            <datatypeField name="inlineText" uuid="1d335e44-05c2-4043-b635-00662dacdc05"/>
         </attribute>
          <attribute name="postalCode" uuid="9fa1242f-73f1-400b-b2ca-aee3b4bb16b9">
             <datatypeField name="inlineText" uuid="1d335e44-05c2-4043-b635-00662dacdc05"/>
          </attribute>
         <attribute name="censusTract" uuid="c8a9bc7e-eaa2-4535-ad7b-c81a3626c72b">
            <datatypeField name="inlineText" uuid="e645170d-6085-4510-bae4-d51dc31d3b02"/>
         </attribute>
         <attribute name="streetName" uuid="e9e29b48-a10a-4940-9535-edb93418f43c">
            <datatypeField name="inlineText" uuid="274b61bb-17d9-4c7d-bf39-f9274ddd256f"/>
         </attribute>
         <attribute name="country" uuid="4bd65115-1392-44b1-be5e-232aff6fe3de">
            <datatypeField name="inlineText" uuid="dfeeae97-1740-4570-9931-8ec17f0a0c11"/>
         </attribute>
      </attribute>
```

```
      <clone clonename="Person" uuid="b684750d-db03-4931-8447-418c9c2d3926">
         <attribute name="templateId" uuid="21a20687-d7aa-4c13-b136-480937cdcc60"/>
         <attribute name="realmCode" uuid="6270166a-2bcc-489c-941b-b8ad68ea5992"/>
         <attribute name="typeID" uuid="f76f6e6e-cdcc-4387-8d9a-dcfd20550b2e"/>
         <attribute name="type" uuid="23f13a13-cfe0-4f3e-9e48-c31fece09261">
            <datatypeField name="code" uuid="2d874a72-fb76-4fe6-bcf7-93ea4c8e6831"/>
            <datatypeField name="displayName" uuid="802d33f7-2049-471a-8b66-3ad68d89b434"/>
            <datatypeField name="codeSystem" uuid="2f2e39df-3784-4b79-a1d5-cdd69e1db52d"/>
            <datatypeField name="codeSystemName" uuid="6b2d3bfa-c548-428d-aa23-2b71774186bc"/>
            <datatypeField name="codeSystemVersion" uuid="c7074b27-d071-4de8-b3fa-
2ac7a487ae22"/>
            <datatypeField name="inlineText" uuid="473e5880-10b4-40c4-9b03-64514bdbfdfa"/>
         </attribute>
         <attribute name="classCode" uuid="9c240763-630d-407d-afd6-cc6eadd92ee7"/>
         <attribute name="determinerCode" uuid="bd289b9a-2ce7-4fd0-87a9-b6c31364f355"/>
         </attribute>
      </clone>
   </clone>
</hl7v3meta>
```

11.3 Function Specification

11.3.1 Overview

The function specification is used as a guide for function objects to read the function specification and determine what objects to call to execute a function (for example, concatenation). The function specification also stores datapoints for rendering by a function graphical representation within the mapping tool. It uses the following types of nested elements:

1. <group name>
2. <function name>
3. <inputs>
4. <datapoint>
5. <outputs>

11.3.2 Function Specification File Example

Following is an example of a function specification file.

```
<functions>
       <group name="string" uuid="c4244890-fe19-11d9-8cd6-0800200c9a66">
              <function name="Concatenate" uuid="bffc6ea0-fe0a-11d9-8cd6-0800200c9a66">
                     <inputs>
                            <datapoint pos="0" name="String1" datatype="string" uuid="e8a43ac0-
fe0c-11d9-8cd6-0800200c9a66"/>
                            <datapoint pos="1" name="String2" datatype="string" uuid="eec52360-
fe0c-11d9-8cd6-0800200c9a66"/>
                     </inputs>
                     <outputs>
                            <datapoint pos="0" name="Result" datatype="string" uuid="f44d9c90-
fe0c-11d9-8cd6-0800200c9a66"/>
                     </outputs>
                     <implementation classname="gov.nih.nci.hl7.function.StringFunction"
method="concat"/>
              </function>
              <function name="Split" uuid="fb31d530-fe0c-11d9-8cd6-0800200c9a66">
                     <inputs>
```

```
                        <datapoint pos="0" name="string1" datatype="string" uuid="041e39e0-
fe0d-11d9-8cd6-0800200c9a66"/>
                        <datapoint pos="1" name="Pos" datatype="int" uuid="041e39e1-fe0d-
11d9-8cd6-0800200c9a66"/>
                    </inputs>
                    <outputs>
                        <datapoint pos="0" name="Result1" datatype="string" uuid="041e39e2-
fe0d-11d9-8cd6-0800200c9a66"/>
                        <datapoint pos="1" name="Result2" datatype="string" uuid="041e60f0-
fe0d-11d9-8cd6-0800200c9a66"/>
                    </outputs>
                    <implementation classname="gov.nih.nci.hl7.function.StringFunction"
method="split"/>
             </function>
             <function name="Substring" uuid="154d6050-fe0e-11d9-8cd6-0800200c9a66">
                    <inputs>
                        <datapoint pos="0" name="String" datatype="string" uuid="154d8760-
fe0e-11d9-8cd6-0800200c9a66"/>
                        <datapoint pos="1" name="StartPos" datatype="string"
uuid="154d8761-fe0e-11d9-8cd6-0800200c9a66"/>
                        <datapoint pos="2" name="EndPos" datatype="string" uuid="8febe8e0-
fe0e-11d9-8cd6-0800200c9a66"/>
                    </inputs>
                    <outputs>
                        <datapoint pos="0" name="Result" datatype="string" uuid="154d8762-
fe0e-11d9-8cd6-0800200c9a66"/>
                    </outputs>
                    <implementation classname="gov.nih.nci.hl7.function.StringFunction"
method="substring"/>
             </function>
             <function name="Trim" uuid="3f47d610-fe0e-11d9-8cd6-0800200c9a66">
                    <inputs>
                        <datapoint pos="0" name="String" datatype="string" uuid="154d8760-
fe0e-11d9-8cd6-0800200c9a66"/>
                    </inputs>
                    <outputs>
                        <datapoint pos="0" name="Result" datatype="string" uuid="154d8762-
fe0e-11d9-8cd6-0800200c9a66"/>
                    </outputs>
                    <implementation classname="gov.nih.nci.hl7.function.StringFunction"
method="trim"/>
             </function>
      </group>
</functions>
```

11.4 Mapping Specification

11.4.1 Overview

A mapping specification has the following main elements:

1. <components>
2. <links>
3. <views>

Components data items used in the file such as a source specification, a target specification or a function specification. Links are associations between a specific element in a source specification and a specific element in a target specification. Views are GUI-related rendering information for displaying items in the mapping tab properly.

11.4.2 Mapping File Example

Following is an example mapping file.

```
<?xml version="1.0"?>

<mapping version="1.2">
    <components>
        <!-- A component is an instance of a particular data. -->
        <component uuid="1111A" type="source" kind="scm"
location="C:\projects\hl7sdk\map\csv\csvspecification.scm"/>
            <!-- when type="scm" the location attribute contains a reference to a file -->

        <component uuid="2222B" type="source" kind="scm" location="csvspecification.scm"/>

            <!-- when type="HL7v3" the value attribute contains a messagetypeid -->
        <component uuid="3333C" type="target" kind="HL7v3" location="NCIC_MT99999.hsm"/>

        <!-- A function component is an algorithm between two (or more) pieces of data. -->
        <component uuid="4444D" type="function" kind="core" group="string" name="concat">
        </component>

        <component uuid="5555E" type="function" kind="core" group="string" name="split">
        </component>

        <component uuid="6666F" type="function" kind="string" name="split">
        </component>
    </components>

    <links>

        <!-- Direct Map -->
        <link uuid="12345A">
            <linkpointer component-uuid="1111A" data-uuid="AAAAAAAAAAAAAAAAAAA"/>
            <linkpointer component-uuid="3333B" data-uuid="1111111128"/>
        </link>

        <!-- Direct Map -->
        <link uuid="98457A">
            <linkpointer component-uuid="1111C" data-uuid="BBBBBBBBBBBBBBBBBBB"/>
            <linkpointer component-uuid="3333D" data-uuid="1111111133"/>
        </link>

        <!-- Funtion Map Inputs-->
        <link uuid="map003">
            <linkpointer component-uuid="1111E" data-uuid="1111ABCDEFGH"/>
            <linkpointer component-uuid="4444F" data-uuid="ABCDEF44444444"/>
        </link>

        <link uuid="map004">
            <linkpointer component-uuid="1111G" data-uuid="1111ABCDEFGH"/>
            <linkpointer component-uuid="4444H" data-uuid="ABCDEF55555555"/>
        </link>

        <!-- Funtion Map Outputs-->
        <link uuid="map005">
            <linkpointer component-uuid="4444I" data-uuid="ABCDEF6666666"/>
            <linkpointer component-uuid="3333J" data-uuid="1111111114"/>
        </link>

    </links>
    <views>
```

```
        <view component-uuid="4444A" x="0" y="0" width="100" height="50" color="black"/>
        <view component-uuid="5555B" x="200" y="200" />
        <view component-uuid="4444C" width="100" height="50" color="black"/>
    </views>
</mapping>
```

11.5 File Extensions

The development team is currently focused on the CSV-to-HL7 mapping, but future development phases need to be kept in mind. *Table 11-1* contains the file extensions that have been identified as standards for development. *Table 11-1* lists the file sources supported in this or a later version of the design and to which the `.map` file points.

File Type	***Extension***
CSV specification	.scs
HL7 v3 specification	.h3s
HL7 v2 specification	.h2s
Database specification	.dbs
Function library specification	.fls
Map specification	.map
HL7 v3 message	.xml

Table 11-1 File extensions

12. Appendix

For consistency, here we list a group of generally used terms that have been referred in context of this document.

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 release, it refers to the CSV Specification
HL7 v3 Message Specification	An XML-based specification that is derived from a Hierarchical Message Descriptions (HMDs) file, 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 release, 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 release, it refers to CSV data file
HL7 v3 Message	The outcome of the transformations service, i.e. the XML messages that comply to HL7 v3 specification
Target Message	In current release, it refers to HL7 v3 Message