Health Level Seven Standard

Context Management ("CCOW") Specification
Technology- and Subject-Independent Component
Architecture Version CM-1.3

DOCUMENT ID: HL7CCOW_2_1_01
REVISION ID: February 25, 2000
FILE NAME: hl7_ccow_arch_cm_1_3.doc
SUPERCEDES: HL7CCOW_6_1_00

Copyright 2001 Health Level Seven
## Contents

1. **INTRODUCTION** .................................................................................................................. 15

1.1 CLINICAL CONTEXT ............................................................................................................ 15

1.2 LINKS AND SUBJECTS ....................................................................................................... 15

1.3 ARCHITECTURE SUMMARY ........................................................................................... 17

1.4 READING THIS DOCUMENT ........................................................................................... 20

2. **SCOPE AND OBJECTIVES** .............................................................................................. 21

2.1 SPECIFICATION ORGANIZATION ................................................................................... 21

2.2 ASSUMPTIONS/ASSERTIONS ............................................................................................ 22

2.3 CMA DESIGN CENTER ..................................................................................................... 24

3. **TECHNOLOGY NEUTRALITY** .......................................................................................... 25

4. **REQUIREMENTS AND CAPABILITIES** ......................................................................... 29

5. **SYSTEM ARCHITECTURE** .............................................................................................. 31

5.1 USE-MODEL ..................................................................................................................... 31

5.2 CONTEXT MANAGEMENT RESPONSIBILITY .................................................................... 41

5.3 CONTEXT CHANGE DETECTION ..................................................................................... 42

5.4 CONTEXT DATA REPRESENTATION ................................................................................ 42

5.5 CONTEXT DATA ACCESS ................................................................................................ 43

5.6 CONTEXT DATA INTERPRETATION ................................................................................ 44

5.6.1 Context Subject Data Definition Constraints ................................................................. 46

5.6.2 Establishing the Meaning of Context Data Item Names ................................................ 46

5.6.3 Establishing the Meaning of Context Data Item Values ................................................ 46

5.6.4 Representing Context Subjects That Cannot Be Uniquely Identified ............................ 47

5.6.5 Context Subjects and Item Name Format ...................................................................... 47

5.6.6 Standard Context Subjects and Data Items .................................................................. 48

5.6.7 Non-Standard Context Subjects and Data Items .......................................................... 48

5.6.8 Representing “Null” Item Values .................................................................................. 48

5.6.9 Representing an Empty Context Subject ....................................................................... 48

5.6.10 Case Sensitivity with Regard to Item Names and Item Values ..................................... 49

6. **COMPONENT MODEL** ................................................................................................... 51

6.1 COMPONENT AND INTERFACE CONCEPTS ................................................................. 52

6.1.1 Interfaces and References ............................................................................................ 52

6.1.2 Interface Interrogation .................................................................................................. 52

6.1.3 Principal Interface ......................................................................................................... 53

6.1.4 Interface Reference Registry ....................................................................................... 53

6.1.5 Interface Reference Management ................................................................................ 53

7. **GENERAL THEORY OF OPERATION FOR CLINICAL LINKS** .................................. 55

7.1 MULTIPLE SUBJECTS AND THE MEANING OF “LINK” .................................................. 55

7.1.1 Context Manager Support for Multiple Context Subjects .............................................. 55

7.1.2 Effect of Multiple Subjects on Context Change Transaction ......................................... 56

7.1.3 Context Manager Treatment of Multi-Subject Context Data ......................................... 57

7.1.4 Effect of Multiple Subjects on Mapping Agents ............................................................... 58

7.1.5 Effect of Multiple Subjects on Annotation Agents ........................................................... 58

7.1.6 Application Treatment of Multiple Subjects .................................................................... 58

7.2 CONTEXT SUBJECTS DEPENDENCIES ......................................................................... 58

7.3 EMPTY CONTEXTS SUBJECTS ...................................................................................... 60

8. **THEORY OF OPERATION FOR COMMON LINKS** ......................................................... 61
# Context Management Specification, Technology and Subject-Independent Component Architecture

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>COMPONENT ARCHITECTURE FOR COMMON LINKS</td>
<td>61</td>
</tr>
<tr>
<td>8.2</td>
<td>MAPPING AGENTS</td>
<td>63</td>
</tr>
<tr>
<td>8.3</td>
<td>ANNOTATION AGENTS</td>
<td>63</td>
</tr>
<tr>
<td>8.4</td>
<td>CONTEXT CHANGE TRANSACTIONS</td>
<td>64</td>
</tr>
<tr>
<td>8.5</td>
<td>JOINING THE COMMON CONTEXT SYSTEM</td>
<td>65</td>
</tr>
<tr>
<td>8.6</td>
<td>CONTEXT CHANGE TRANSACTIONS</td>
<td>65</td>
</tr>
<tr>
<td>8.7</td>
<td>TRANSACTIONAL CONSISTENCY</td>
<td>66</td>
</tr>
<tr>
<td>8.8</td>
<td>CONTEXT CHANGE NOTIFICATION PROCESS</td>
<td>66</td>
</tr>
<tr>
<td>8.9</td>
<td>LEAVING A COMMON CONTEXT SYSTEM</td>
<td>69</td>
</tr>
<tr>
<td>8.10</td>
<td>BEHAVIORAL DETAILS</td>
<td>69</td>
</tr>
<tr>
<td>8.11</td>
<td>COMMON CLINICAL CONTEXT USE MODEL</td>
<td>72</td>
</tr>
<tr>
<td>9.1</td>
<td>MAPPING AGENTS</td>
<td>91</td>
</tr>
<tr>
<td>9.2</td>
<td>ASSUMPTIONS AND ASSERTIONS</td>
<td>92</td>
</tr>
<tr>
<td>9.3</td>
<td>THEORY OF OPERATION</td>
<td>92</td>
</tr>
<tr>
<td>9.4</td>
<td>MAPPING AGENT EFFECT ON APPLICATION SECURITY POLICIES</td>
<td>94</td>
</tr>
<tr>
<td>9.5</td>
<td>Distinguishing Between Mapping Agents and Context Participants</td>
<td>94</td>
</tr>
<tr>
<td>9.6</td>
<td>Mapping Agent Updates to Context Data</td>
<td>94</td>
</tr>
<tr>
<td>9.7</td>
<td>Conditions for Mapping Agent Invalidation of Context Changes</td>
<td>98</td>
</tr>
<tr>
<td>9.8</td>
<td>Treatment of Mapping Agent Invalidation of Context Changes</td>
<td>98</td>
</tr>
<tr>
<td>9.9</td>
<td>Mapping Null-Value Identifiers</td>
<td>100</td>
</tr>
<tr>
<td>9.10</td>
<td>Initializing Mapping Agents</td>
<td>101</td>
</tr>
<tr>
<td>9.11</td>
<td>Handling Mapping Agent Failures</td>
<td>102</td>
</tr>
<tr>
<td>9.12</td>
<td>MAPPING AGENT EFFECT ON APPLICATION SECURITY POLICIES</td>
<td>103</td>
</tr>
<tr>
<td>9.13</td>
<td>Secure Annotation Agent</td>
<td>113</td>
</tr>
<tr>
<td>9.14</td>
<td>Context Management Interfaces</td>
<td>113</td>
</tr>
<tr>
<td>9.15</td>
<td>Overall Secure Link Component Architecture</td>
<td>114</td>
</tr>
<tr>
<td>9.16</td>
<td>PROCESS FOR SETTING A Secure Context Subject</td>
<td>115</td>
</tr>
<tr>
<td>10.1</td>
<td>ANNOTATION AGENTS</td>
<td>107</td>
</tr>
<tr>
<td>10.2</td>
<td>ASSUMPTIONS AND ASSERTIONS</td>
<td>107</td>
</tr>
<tr>
<td>10.3</td>
<td>THEORY OF OPERATION</td>
<td>108</td>
</tr>
<tr>
<td>11.1</td>
<td>THEORIES OF OPERATION FOR SECURE LINKS</td>
<td>111</td>
</tr>
<tr>
<td>11.2</td>
<td>SECURE LINK TERMS</td>
<td>111</td>
</tr>
<tr>
<td>11.3</td>
<td>DESKTOP ASSUMPTIONS</td>
<td>112</td>
</tr>
<tr>
<td>11.4</td>
<td>SECURITY LINK COMMON CONTEXT SYSTEM DESCRIPTION</td>
<td>112</td>
</tr>
<tr>
<td>11.5</td>
<td>Authentication Secure Subject System</td>
<td>113</td>
</tr>
<tr>
<td>11.6</td>
<td>Secure Mapping Agent</td>
<td>113</td>
</tr>
<tr>
<td>11.7</td>
<td>Secure Annotation Agent</td>
<td>113</td>
</tr>
<tr>
<td>11.8</td>
<td>Context Management Interfaces</td>
<td>113</td>
</tr>
<tr>
<td>11.9</td>
<td>Overall Secure Link Component Architecture</td>
<td>114</td>
</tr>
<tr>
<td>11.10</td>
<td>PROCESS FOR SETTING A Secure Context Subject</td>
<td>115</td>
</tr>
</tbody>
</table>

4 Copyright 2001, Health Level Seven  Version CM-1.3
Context Management Specification, Technology and Subject-Independent Component Architecture

[5.3.6.4] ContextChangesCanceled
[5.3.6.5] CommonContextTerminated
[5.3.6.6] Ping

[5.3.7.1] Synopsis

[5.3.7.2] ComponentName

[5.3.7.3] RevMajorNum

[5.3.7.4] RevMinorNum

[5.3.7.5] PartNumber

[5.3.7.6] Manufacturer

[5.3.7.7] TargetOS

[5.3.7.8] TargetOsRev

[5.3.7.9] WhenInstalled

[5.3.8] MappingAgent (MA)

[5.3.8.1] Synopsis

[5.3.8.2] ContextChangesPending

[5.3.8.3] Ping

[5.3.9] SecureBinding (SB)

[5.3.9.1] Synopsis

[5.3.9.2] InitializeBinding

[5.3.9.3] FinalizeBinding

[5.3.10] SecureContextData (SD)

[5.3.10.1] Synopsis

[5.3.10.2] GetItemNames

[5.3.10.3] SetItemValues

[5.3.10.4] GetItemValues

[16] CONFORMANCE

[16.1] General Approach and Goals

[16.1.1] Context Management Standard Declaration of Conformance

[16.1.2] Future Conformance Requirements

[16.2] Applications

[16.3] Context Manager

[16.4] Mapping Agent

[16.5] Annotation Agent

[16.6] Authentication Repository

[17] Backwards Compatibility

Appendix: Diagramming Conventions

Glossary

Figures

Figure 1: Patient Linked Applications

Figure 2: Organization of HL7 Context Management Specification Documents

Figure 3: Overall Role of the CMA Specification

Figure 4: COM/Java/CORBA Interoperability

Figure 5: Patient Selection Change Use Case

Figure 6: Patient Context Automatically Changes within all Context Participant Applications
Figure 7: User Informed of Potential Data Loss and Cancels Context Change................................. 37
Figure 8: User forces Application AAA to Become Out of Synchrony with other Context Participants 38
Figure 9: Context Participant Not Responding to Selection Change Request.................................. 39
Figure 10: User Accepts Consequences of going ahead with Patient Selection Change with all Applications ................................................................................................................ 40
Figure 11: Component Architecture for Common Links ..................................................................... 62
Figure 12: Context Change Process for Common Links ..................................................................... 64
Figure 13: Common Clinical Context Use Model .................................................................................. 72
Figure 14: Common Context Lifecycle Use Case .................................................................................. 73
Figure 15: Context Selection Change Use Case .................................................................................... 76
Figure 16: Abnormal Termination of Common Context Use Case ........................................................ 85
Figure 17: Component Architecture for Secure Links .......................................................................... 114
Figure 18: User Link Sign-On Process ............................................................................................... 115
Figure 19: User Link Component Architecture .................................................................................... 125
Figure 20: User Link Context Change Process ..................................................................................... 126
Figure 21: User Subject Context Data Mapped for Different Applications ........................................... 134
Figure 22: Signing A Message ............................................................................................................. 146
Figure 23: Forming Signature Using Method Parameters ........................................................................ 147

Tables

Table 1: User Link-Enabled Application Behavior for Termination and Log-Off .................................... 129
Table 2: Chain of Trust Attacks and Defenses ..................................................................................... 142
Table 3: Handling Transaction Instigator Failure .................................................................................. 169
Table 4: Character Representations for Basic Data Types ..................................................................... 171

Interaction Diagrams

Interaction Diagram 1: Common Context Lifecycle ............................................................................... 74
Interaction Diagram 2: Suspending/Resuming Context Participation ...................................................... 75
Interaction Diagram 3: All applications accept the changes ..................................................................... 77
Interaction Diagram 4: An application conditionally accepts the changes; user decides to cancel changes .............................................................................................................................. 78

Interaction Diagram 5: An application does not respond to survey .................................................................................................................. 79

Interaction Diagram 6: An application does not respond to change notification .......................................................................................... 80

Interaction Diagram 7: An application responds after context change transaction has completed ........................................................................ 81

Interaction Diagram 8: A non-surveyed application participates in context change ......................................................................................... 82

Interaction Diagram 9: An application conditionally accepts the changes; user decides to accept consequences of change .............................................................................................................................. 83

Interaction Diagram 10: An application conditionally accepts the changes; user breaks link with common context .............................................................................................................................. 84

Interaction Diagram 11: Abnormal Termination of Common Context .............................................................................................................. 86

Interaction Diagram 12: Simplest Application .................................................................................................................................................. 89

Interaction Diagram 13: Context Change Transaction with Mapping Agent .................................................................................................. 97

Interaction Diagram 14: Mapping Agent Invalidates Context Change Transaction ................................................................................................. 102

Interaction Diagram 15: User Logs Off From One Application ......................................................................................................................... 131

Interaction Diagram 16: User Logs-Off From Desktop ............................................................................................................................... 131

Interaction Diagram 17: Populating Authentication Repository with User Authentication Data ........................................................................... 159

Interaction Diagram 18: User Link Context Change Transaction .................................................................................................................. 160
Preface

This document was prepared by Robert Seliger, Sentillion, Inc., on behalf of Health Level Seven’s CCOW Technical Committee. Comments about the organization or wording of the document should be directed to the author (robs@sentillion.com). Comments about technical content should be directed to ccow@lists.hl7.org.
Changes from 1.2

- Added specification for annotation subjects, a new category of context data items.
- Added specification for an annotation agent, a component for populating an annotation subject with data during a context change transaction.
- Added specification for the interface ContextAgent, which is implemented by an annotation agent, and may also be used by mapping agents as an optimized alternative to the existing MappingAgent interface.
- Enabled the currently unused inputs participantCoupon and appSignature, defined for the method SecureContextData::GetItemValues, so that the context manager has the capability to enforce access privileges when applications attempt to get context data. In so doing, also expanded the types of subject-level access privileges, as returned in the output parameter privileges.
- Removed the chapters describing Encounter Link and Custom Links as they did not add any value beyond what could be found in the Data Definitions specification document.
1 Introduction

This document specifies the Health Level Seven Context Management Architecture (CMA). This architecture enables multiple applications to be automatically coordinated and synchronized in clinically meaningful ways at the point-of-use. The architecture specified in this document establishes the basis for bringing interoperability among healthcare applications to the point-of-use, such as the clinical desktop.

1.1 Clinical Context

Clinical context is state information that a user establishes and modifies while interacting with healthcare applications at the point-of-use (e.g., a clinical desktop). The context is common because it establishes parameters that should uniformly affect the behavior or operation of multiple healthcare applications. The context needs to be managed so that the user has a way of controlling it, and so that applications have a way of robustly coordinating their behavior as the context changes.

Examples of clinical context include but are not limited to:

- The identity of a patient whose data the user wants to view or update via the applications.
- The identity of the user who wants to access the applications.
- A moment in time around which temporal data displays should be centered by the applications.
- A particular patient encounter that the user wants to review via the applications.

Healthcare application developers often implement a common clinical context capability for their own applications. However, there are currently no standards that enable independently-developed applications to share a common clinical context. Further, with the diversity of application programming technologies currently available, a common context solution should strive to be applicable to at least several of the dominant and emerging technologies.

1.2 Links and Subjects

The approach taken for the CMA is to define the architecture that enables applications to establish a single link based upon a set of clinical subjects of common interest. The applications automatically and cooperatively change their state whenever the user sets a new value for one or more of these subjects. While the complete set of subject data definitions are
specified in the document Health Level-Seven Standard Context Management Specification, Subject Data Definitions, Version CM-1.3, there are two standard link subjects that are core to the CMA, and are therefore introduced in this document:

- **Patient**, which enables the user to select the patient of interest once from any application as the means to automatically “tune” all of the applications to the selected patient.

- **User**, which enables the user to securely logon once to any application as the means to automatically “tune” all of the applications to the user.

In addition, healthcare vendor and/or provider organizations can define custom subjects and thereby implement custom links between applications that understand these subjects. The basis for custom subjects is also established in this document.

Applications that share the same common context are said to comprise a *common context system*. These applications have established and maintain a common context link. There is only one link, while there can be multiple subjects. However, in the vernacular that arose as the CMA was being developed, it became useful to refer specific link subjects. This has given rise to the terms such as *Patient Link* and *User Link*. An example of a set of Patient Linked applications is shown in Figure 1.

![Figure 1: Patient Linked Applications](image)
1.3 Architecture Summary

The CMA defines the interfaces between applications, known as context participants, and a coordinating component, known as the context manager. The CMA also defines the policies that govern the use of these interfaces and the interaction among and between CMA-compliant applications and components.

Applications that share a common context with each other, and the context manager that mediates the applications, are collectively referred to as a common context system. Applications only need to interact with the context manager in order to participate in a common context system.

The data that represents the common clinical context for a common context system resides in the context manager. The data is organized as a set of data items that are grouped by context subject. There are two classes of context subjects:

- An identity subject identifies a real-world entity (e.g., a specific patient) or concept (e.g., a specific encounter).
- An annotation subject contains data that describes or is otherwise pertinent to the entity or concept identified in an identity subject.

Identity subjects may be defined such that they are dependent upon each other. It is necessary to define identity subjects as dependent when the identifier value set for one subject must be consistent with the identifier value set for another subject. An example is the encounter subject, which is dependent upon the patient subject. This is because an encounter is patient-specific. The encounter identifier value, if set, must be consistent with the identifier value set for the patient subject.

Annotation subjects are always dependent upon an identity subject. This is because an annotation subject does not identify anything, but rather provides additional information pertaining to an entity or concept identified by an identity subject. Each annotation subject contains additional context data items that annotates a context subject with descriptive information. For example, the patient context subject identifies the current patient, but a hypothetical demographics annotation subject might annotate the patient subject with data that contains the patient’s demographic information. The demographics data, if set, must be consistent with the identifier value set for the patient subject.

A data item within a context subject is represented as a name/value pair. For an identity subject at least one of these items is the identifier for the real-world entity or concept represented by the subject.
In order for applications to have a common and unambiguous understanding of the semantic meaning for each data item that comprises a context subject, the data definitions for these items must be known when the applications are developed. CCOW has therefore defined a set of standard subjects such as patient, user, and encounter, and for each subject a corresponding set of standard data items. Additionally, individual organizations or groups of organizations may define custom subjects that follow data definitions defined by the organization(s).

In addition to defining the data definitions for each context subject's data items, the definition of a context subject also indicates whether or not the subject is a secure subject. When a subject is defined as secure, only applications with the appropriate access privileges are allowed by the context manager to set and/or get the context data for the subject. The necessary application privileges are configured on a site-specific basis. When a subject is not secure, application access privileges are not enforced.

Applications must identify themselves using an application-specific digital signature when they interact with the context manager to access a secure subject. This provides the context manager with a reliable means for identifying applications in order to enforce their access privileges. A digital signature can be readily authenticated but not easily violated.

Similarly, the context manager identifies itself to applications using its own digital signature. This enables applications to be certain that they are interacting with the real context manager, as opposed to a rogue component. This overall scheme of bilateral authentication is referred to as the “chain of trust.”

When the user performs an application gesture that instructs the application to change the common clinical context (e.g., the user has selected a different patient), the application starts a context change transaction. Context items can be added or removed, or have their values set, during a context change transaction. Only one transaction can be in progress at a time.

When the application that instigated the transaction has completed its changes to the context data, the context manager conducts a two-phase process to coordinate the propagation of the context changes to the other applications.

In the first phase, the context manager surveys the other applications to determine which ones can apply the new context, and which ones either cannot, or prefer not to. An application cannot apply the changes if it is blocked, for example if it is waiting for the user to enter data. An application might prefer not to apply the new context if, for example, doing so might cause the user to lose work-in-progress.

The context manager informs the instigating application of the survey results. If all of the applications are willing to apply the new context, then they are all instructed to do so. If at least one of the surveyed applications is blocked (“busy”) or prefers to keep the previous context, then the user is asked by the instigating application to decide how to proceed:
- The user can cancel the context change.
- The user can break the link between the instigating application and the other applications. The new context is then applied only to the instigating application, while the other applications remain linked together and tuned to the previous context.
- The user can apply the changes anyway (as long as there are no busy applications).

The context manager broadcasts the decision to all of the context participants to complete the second phase of the transaction. This approach ensures that the link among application is never broken unless the user has performed an explicit gesture instructing that the link be broken.

Context agents are a class of CMA components that provide an automatic means for adding data to the context during a context change transaction. This data is in addition to the data that is set by the application that instigate a context change transaction.

Mapping agents are a type of context agent that provides an automatic means for adding identifier items to an identity subject. For its subject, a mapping agent is responsible for computing a mapping between the identifier items set by the application that instigated a context change transaction and additional identifier items that also represent the subject. The additional identifier items enable all of the participant applications to “tune” to the same subject even when they do not necessarily have a common way to identify the subject.

A mapping agent only interacts with the context manager, so its existence is transparent to the applications. Prior to the second phase of a context change transaction, the context manager allows each mapping agent that is present to add data to the identity subject it is responsible for mapping.

There can be a mapping agent for each subject. For its subject, a mapping agent is responsible for obtaining the mapping data from an authentic source within the enterprise. For example, patient mapping agent might obtain its data from an enterprise’s master patient index system. The means by which a mapping agent obtains data from an authentic enterprise source is not specified by the CMA.

Annotation agents are a type of context agent that provides an automatic means for setting all of the data that comprises a particular annotation subject. Prior to the second phase of a context change transaction, but after the mappings performed by the mapping agents, the context manager allows each annotation agent that is present to set the data for an annotation subject.

For its subject, an annotation agent is responsible for obtaining this data from an authentic source within the enterprise. For example, the hypothetical demographics annotation agent
might obtain its data from an enterprise’s patient registration system. There may be an
annotation agent for each annotation subject.

Like a mapping agent, an annotation agent only interacts with the context manager, so its
existence is transparent to the applications. The means by which an annotation agent obtains
data from an authentic enterprise source is not specified by the CMA.

1.4 Reading This Document

This document presents a comprehensive specification of the HL7 Context Management
Architecture. The precision of the specification becomes increasingly more detailed as the
document progresses. Several of the early chapters present concepts that underlie the
architecture and lead the reader through the rationale for various architectural choices, while
all of the chapters in this document include information that the reader should find pertinent
to the explanation of the CMA.

However, Chapters 5 through 16 all contain normative content and as such should be
regarded as the core of the CMA specification. In particular, Chapter 15, Interface
Definitions, concludes the core specification with the complete set of CMA interface
definitions, including methods and their argument signatures. These interfaces are ultimately
the basis for the implementation of applications and components that conform to the CMA
specification.

A compliant CMA application or component shall implement the relevant set of CMA
interfaces exactly as specified. A compliant application or component implementation shall
adhere to these interface definitions and to the policies specified throughout this document
that govern the use and behavior of these interfaces. See Chapter 16 for a complete
description of what is required for applications and components to be able to claim
conformance.
2 Scope and Objectives

The HL7 Context Management Architecture (CMA) enables independently developed applications to share data that describes a common clinical context. This document emphasizes the policies, protocols, software interfaces, and responsibilities applications must implement and adhere to as participants in a shared context system.

A common context system is comprised of applications launched directly or indirectly by a particular clinical end-user, wherein the applications share the same context data. Also included in this system is a context management facility that enables applications to share the context data.

2.1 Specification Organization

It is beyond the scope of this document to provide all of the details that are needed in order to fully implement a conformant CMA system. The necessary additional details are covered in a series of companion specification documents. As illustrated in Figure 2, these documents are organized to facilitate the process of defining additional link subjects and to accelerate the process of realizing the CMA using any one of a variety of technologies.

Figure 2: Organization of HL7 Context Management Specification Documents

The context management subjects and technologies that are of interest are determined by the HL7 constituency. There is a single HL7 context management data definition specification.
document for all of the standard link subjects. This document defines the data elements that  
comprise each link subject. Concurrent with the publication of this document, the following  
document has been developed:

Health Level-Seven Standard Context Management Specification,  
Subject Data Definitions, Version CM-1.3

There is an HL7 context management user interface specification document for each of the  
user interface technologies with which CMA-enabled applications can be implemented. Each  
document reflects the user interface requirements established in this document in terms of a  
technology-specific look-and-feel. Concurrent with the publication of this document, the  
following document has been developed:

Health Level-Seven Standard Context Management Specification,  
User Interface: Microsoft Windows and Web Browsers, Version CM-1.3

Finally, there is an HL7 context management component technology mapping specification  
document for each of the component technologies that can be used to implement the CMA.  
Each document provides the technology-specific details needed to implement CMA-  
compliant applications and the associated CMA components, as specified in this document.  
Concurrent with the publication of this document, the following documents have been  
developed:

Health Level-Seven Standard Context Management Specification,  
Component Technology Mapping: ActiveX, Version CM-1.3

Health Level-Seven Standard Context Management Specification,  
Component Technology Mapping: Web, Version CM-1.3

2.2 Assumptions/Assertions

Key assertions and assumptions that were made during the course of developing the CMA  
are indicated below:

- The architecture does not intend to solve nor is it a substitute for solving the patient  
identification problem. However, the architecture does attempt to accommodate

---

1 In general, patients cannot be reliably identified using their given name because given names are not  
necessarily unique. Identifiers can be assigned, but often a single person accumulates multiple patient  
identifiers over time. This is because the assigned identifiers are not universally unique, and generally  
only refer to a population of patients known to a particular healthcare institution, or known to a site  
within an institution. Government assigned identifiers, such as a social security number, may not be  
unique, or may change over time. In general, there is currently no simple and reliable way to identify  
the same patient across all possible systems that might contain data pertinent to the patient.
established means for achieving consistent interpretations of patient identification information.

- Architectural support for context data other than that which is used to identify patients is a non-objective to the extent it complicates the architecture. However, the architecture is currently applicable to a wide range of context data elements.

- Architectural support for distributed applications is a non-objective to the extent it complicates the architecture. However, the architecture is currently applicable to distributed as well as co-located applications.

- Context management is not a form of data interchange nor is it a substitute for data interchange. However, the common context might contain data that can also be obtained by an application through data interchange mechanisms such as those based upon HL7 (e.g., a patient’s name or date of birth in addition to a patient identifier). When such data is provided, it is only as a means to simplify or optimize the sharing of common context.

- The context management facility is not visible to the clinical end-user. However, it might be visible to a systems integrator or systems administrator.

- The architecture is intended for use in clinical systems that are configured by an IT staff. Ad-hoc installation and configuration of a common context system by the clinical user is a non-objective to the extent it complicates the architecture.

- There is at most one context management facility per clinical desktop. However, applications shall work correctly with any facility implementation that conforms with the CMA specification. It is the decision of the IT staff as to which facility implementation is actually used by a clinical system.

- Implementation complexities will be shifted to the context management facility, as opposed to the applications, whenever this tactic is practical and reasonable. Minimizing the burden for the application developer is valued as an essential element for attracting the participation of the widest possible array of applications.

- It is assumed that the clinical data used by applications that share a common clinical context are appropriately synchronized (e.g., via back-end data interchange) to the degree necessary to ensure the consistent interpretation of the common context.

- It is assumed that any application that has been activated by the user can be used to set the user’s common clinical context as long as the application conforms to the CMA specification. This enables multiple applications to provide context setting capabilities, which is convenient for the user.
• It is assumed that any application that does not understand or is otherwise unable or unwilling (e.g., for security reasons) to respond to a change in the common clinical context will ignore the change. However, any application that chooses to ignore a context change must clearly indicate its decision, for example by blanking its data display and/or minimizing itself.

2.3 CMA Design Center

The CMA specification is primarily aimed at enabling interoperability in the form of application control by the end user. Applications that interoperate in this manner appear to the user as visually integrated. This is because the user can see ways in which the applications interoperate.

This is in contrast to traditional healthcare standards, which have been primarily aimed at enabling interoperability in the form of data interchange between applications. Further, the design focus for the CMA specification is applications that have a means for interchanging clinical data. The overall role of the CMA specification is illustrated in Figure 3.

Figure 3: Overall Role of the CMA Specification
3 Technology Neutrality

As recently as one year ago, it would have sufficed to architect and implement a common clinical context solution that was targeted specifically for the Microsoft Window platforms. With the recent explosion of Web-based technologies, such as Java, this restriction is no longer practical. Fortunately, it is possible to architect a solution that is not predicated upon a specific technology. Specifically, in the architecture described in this document, the concept of technology neutrality is also applied.

The term “technology neutral” does not mean that any technology is applicable. Rather, it means that the common clinical context approach should work equally well with any one of a candidate set of relevant technologies.

The candidate technologies considered for this document are based upon market leadership:

- Inter-component communication: via Microsoft Automation through COM/DCOM; via any CORBA 2.0 compliant object request broker.
- Programming languages: any language that can be interfaced with Microsoft Automation and/or CORBA (e.g., VisualBasic®, C++, Java, MUMPS).
- Operating Systems: Windows 95®; Windows NT®; any platform that can host a Java virtual machine.

The primary reason that technology neutrality is practical is because all of these technologies have a lot in common, including:

- They are all based upon object-oriented principles.
- They are all embraced by Microsoft or are readily available on Microsoft platforms.

These two points have an interesting consequence: the technologies are compatible and interoperable. This makes it a lot easier to be technology neutral. For example:

- CORBA supports multiple programming languages. Support already exists for C, C++, Smalltalk, Java, and MUMPS. Objects implemented in any of these languages can transparently interoperate using CORBA.
- COM supports multiple programming languages. Support already exists for C++, VisualBasic, ObjectPascal, Java, and MUMPS. Objects implemented in any of these languages can transparently interoperate using COM.
- Most vendor’s CORBA object request brokers enable CORBA objects to transparently interoperate with COM objects.
• Microsoft’s Java virtual machine enables Java objects (applets) to transparently interoperate with COM objects.

• Java objects (applets) can transparently communicate with remote Java objects using the Java Remote Method Invocation (RMI) mechanism.

Given the synergistic state of the dominant object technologies, the emphasis of this document is on the structure of the common context system, the roles and responsibilities of the components that comprise the system, the precise definition of the interfaces they need to implement in order to be participants, the interactions between the components (via their interfaces), and a host of architectural decisions that are intended to result in a robust, practical, and useful common context solution.

Figure 4 illustrates a COM-encapsulated Java object that interoperates with other COM objects, and C++ and Java CORBA objects that interoperates with other CORBA objects.
Figure 4: COM/Java/CORBA Interoperability
4 Requirements and Capabilities

The architecture described in this document is intended to serve as an extensible basis for future, more advanced, common clinical context capabilities. However, for now, an attempt will be made to focus on the immediate issue of developing a robust solution for sharing a common patient selection context.

In a complete solution, at least the following issues need to be addressed:

- Extensibility - how can new context elements be easily added in the future?
- Coordination - how can applications be coordinated so that they respond to context setting changes in an orchestrated and manageable manner?
- Flexibility - how can applications and common context managers be structured so that they implement only the capabilities that they need?
- Performance - how can applications and common context managers be structured so that their temporal performance and utilization of computing resources is acceptable to the end-user?
- Localizability - how are internationalization issues addressed (e.g., local character sets, etc.)?
- Scalability - how is the performance of a common context system affected by the quantity of active applications?
- Applicability - how should context information be structured and managed so that application behaviors are useful to the end user?
- Usability - what are the policies that govern the use of a common context such that the resulting application behaviors are intuitive and reasonable?
- Verifiability - how will the correctness of independently developed common context implementations be verified?

Architectural approaches that address these issues are presented next.
5 System Architecture

At the most abstract level, the Context Management Architecture (CMA) provides a way for independent applications to share data that describe a common clinical context. However, the CMA must provide solutions for the following problems:

• What is the general use model for a common context, from the user’s perspective?

• Where does the responsibility for context management reside?

• How are changes to context data detected by applications?

• How is context data organized and represented so that it can be uniformly understood by applications?

• How is context data accessed by applications?

• How is the meaning of context data consistently interpreted by applications?

Before drilling into the details of the complete CMA, this chapter presents approaches and associated trade-offs for the problems listed above.

5.1 Use-Model

There are many possible use-models for a common clinical context.

The extremes of application support for making context changes are represented by:

• Context changes can be performed only via a single, distinguished, application.

• Context changes can be performed via any application.

In the model chosen for the CMA, context changes can be performed via any application. This is because it is not reasonable to assume the universal existence of a distinguished application, and it is beyond the interests and scope of HL7 to specify one.

The extremes of application behavior when context changes are made are represented by:

• When the user changes the context, the changes are automatically communicated to all of the applications that share the context. Applications that are able and willing to apply the context changes do so immediately. Applications that are unable or unwilling to apply the context changes maintain their current context. It is assumed that the user can easily determine which context an application is using.
- When the user changes the context, the changes are automatically communicated to all of the applications that share the context. However, the context changes are only allowed if all of the applications are able and willing to apply the context changes immediately.

The model developed for the CMA is a hybrid of these two extremes that attempts to enable a high degree of automatic context management while also emphasizing clinical safety:

- The likelihood that applications can become uncoordinated with regard to a common clinical context is minimized.

- The circumstances that can prevent context changes from being automatically applied are expected to be infrequent.

The CMA model also respects the challenges of retrofitting common context capabilities into existing healthcare applications. Only modest assumptions about the capabilities of these applications and technology used to develop them are presumed. The CMA model is as follows:

- All or part of the common context can be set by the user from any application for which providing this capability is functionally relevant.

- When the user changes the context, the change is automatically communicated to all of the applications that share the context. The applications are expected to apply the new context in a clinically meaningful manner. In general, applications are also expected to apply the context changes immediately. Exceptions are described below.

- An application may choose to defer applying a context change until some time in the future. For example, an application that retrieves large medical image files (that require substantial processing) might choose to not retrieve images each time a different patient is selected as part of the clinical context. Instead, the application might wait for an explicit directive or gesture from the user before actually retrieving the image. An application that behaves in this manner must be sure that it does not show data for an earlier context. Blank-out its data displays or minimizing itself are possible ways that this can be accomplished.

- An application for which a change in the context might result in the loss of work performed by the user can request that the user explicitly decide whether to proceed with the context change anyway, or to cancel the change. The solicitation of user input is performed by the application that is being used to change the context. The solicitation includes an identification of the application for which work might be lost and a description of the work that might be lost. An application that behaves in this manner is expected to be able to discard its work in progress and apply the context changes if instructed to do so. For example, a medication ordering application might
indicate that the inputs for a medication order that has not yet been completed by the user will be lost if the context is changed to a different patient.

- When an application is unable to respond to a context change, perhaps because the user left it waiting for user input, the user is asked to explicitly decide about how to proceed. The solicitation of user input is performed by the application that is being used to change the context. The solicitation includes the identification of the non-responsive application and indicates that the application cannot respond to a context change. For patient safety reasons, when there are applications that cannot respond to the changes, context changes will not be automatically applied to the applications that share a common context.

- When it is not desirable or possible for context changes to be automatically applied, either because there are applications for which work might be lost, or there are busy applications that cannot be notified about context changes, the user can explicitly interact with these applications to correct the situation, and then apply the context changes. For example, the user might complete or terminate a dialog that was left open in order to enable an application to apply the context changes.

- When it is not desirable or possible for context changes to be automatically applied, the user can also decide to apply the context change only to the application that is being used to change the context. The decision to do this is typically in response to an interruption during which the user needs to momentarily divert her attention to a different context for a specific application. The application is, in effect, disconnected from the common context, and must clearly indicate this fact to the user in a visual manner. The application can be subsequently instructed by the user to reconnect and apply the common context. The common context may have changed between the time the application was disconnected and the time it is reconnected to the common context.

A high-level summary of the interactions between applications when a clinical patient context is changed is illustrated below. Figure 5 illustrates the use case actors (i.e. external forces) involved in a context change such as a patient selection. (The actors are the user plus applications, all of which are represented in the Jacobson modeling technique as stick figures.) Figure 6 through Figure 10 illustrate some possible instances of the Patient Selection Change Use Case from the user’s perspective. Not all possible instances of this use case are provided.
The initial condition for each of the use case instances is that the currently selected patient is Jane Doe. In each instance, the user changes the common clinical context by selecting the patient Sam Smith. Some possible alternative outcomes follow:

- **Figure 6** illustrates all applications reacting to the context change by changing their context to the patient “Sam Smith.”

- **Figure 7** illustrates an application (Application DDD) conditionally accepting the context change and providing information describing work that could be lost if a context change occurs at this time. The user deciding to cancel the change is shown.

- **Figure 8** illustrates a use case instance similar to **Figure 7**. However, the possible outcome of the user deciding to force a context change within Application AAA while the other applications remain with the original context is shown. This exemplifies Application AAA disconnecting from the common context system. Once disconnected, Application AAA’s context is no longer in synchrony with the other applications.

- **Figure 9** illustrates healthcare application DDD not responding to a selection change request in a timely fashion. The user deciding to cancel the change is shown.
• Figure 10 illustrates the user being notified of potential data loss if selection change proceeds. The user accepting these consequences and proceeding with the change is shown.
Figure 6: Patient Context Automatically Changes within all Context Participant Applications
Figure 7: User Informed of Potential Data Loss and Cancels Context Change
Figure 8: User forces Application AAA to Become Out of Synchrony with other Context Participants
Figure 9: Context Participant Not Responding to Selection Change Request
User

Application AAA

Application BBB

Application CCC

Application DDD

Selected patient is "Jane Doe"

I choose "Sam Smith"

select "Sam Smith"

select "Sam Smith"

select "Sam Smith"

can I change selection?

change accepted

change accepted

Application DDD warns "You could lose work."

conditionally accept: "You could lose work."

Go ahead with selection change

accept selection change

accept selection change

accept selection change

Selected patient is "Sam Smith"

Figure 10: User Accepts Consequences of going ahead with Patient Selection Change with all Applications
5.2 Context Management Responsibility

There are two fundamental schemes for architecting the responsibility for context management:

- **Distributed**: The responsibility for managing the common context is uniformly distributed among the applications. There is no central point of common context management.

- **Centralized**: The responsibility for managing the common context is centralized in a common facility that is responsible for coordinating the sharing of the context among the applications.

In the distributed model, applications must either all know about each other, or at least form a completely connected graph within which each application knows at least one other application. This is necessary in order for the applications to communicate context and control data among themselves.

Further, each application has the responsibility to act as a server for the common context in addition to acting as a client of the context. This is to offset the fact that there is no central point of ownership for the context, so each application must be capable of being an owner. This may be elegant, but it does introduce implementation complexities and burdens on all applications.

In the centralized model, applications only need to know about a common service or resource. This service off-loads from the applications much of the burden of maintaining and managing the common context. While a centralized service represents a single point of failure and a potential performance bottleneck, it is nevertheless the approach that is pursued in this document. The primary reasons include:

- It is simpler from the perspective of the application developer.

- The consequence of the service being a single point of failure is offset by the fact that the service and the applications it serves are typically co-resident on the same personal computer. Failures, if any, will be localized to a single user.

- The consequence of the service being a performance bottleneck is offset by the fact that the applications are far more likely to become the performance bottlenecks.

Given this basic system structure, the approaches for the other major architectural issues are summarized next.
5.3 Context Change Detection

There are at least two distinct categories of architectural approaches for realizing a common clinical context system:

- **Pull-model**: A shared component is used to maintain the shared context data. Applications update this resource to change the data. Other applications periodically poll the component to determine if the data has changed.

- **Push-model**: A shared component is used to maintain the shared context data. This component notifies applications whenever the data is changed. In order to receive a notification, an application must have first explicitly indicated its interest in being notified.

Both models have advantages and disadvantages. For example, the pull model is simpler to implement (e.g., does not require applications to handle asynchronous notifications), but can lead to performance problems due to polling even when the context data has not changed. Conversely, the push model can be the basis for better performance, but introduces additional implementation complexity. Both models introduce the additional challenges of synchronizing concurrent access to the context data (e.g., to prevent two applications from attempting to change the data at the same time). In addition, both models must deal with failures modes that can occur when independent applications (i.e., applications that may be implemented as separate executables) are involved. For example, an application that crashes in the middle of changing the context data may leave the context data in an inconsistent state.

Given this analysis, the approach that is taken for the CMA is perhaps best described as a robust push-model. This is a push model that deals with synchronization and partial failure issues.

5.4 Context Data Representation

There are at least three distinct categories of architectural approaches for representing the common context data:

- **Fully-populated objects**: Objects are defined with properties and methods that model the real-world entities that they represent (e.g., a patient, a provider, etc.). These objects may be complex and involve a rich structure (e.g., are comprised of a logical network of objects).

- **Fully-populated messages**: Messages (as in “HL7 messages”) are used to convey detailed information about the context data.

- **Name-value pairs**: A set of name-value pairs represent only key summary information about the common context (e.g., just the patient’s name and medical
The symbolic name for an item describes its meaning. The data types for the items come from a set of simple primitive data types.

The fully-populated object approach is perhaps the purest approach, but is subject to performance concerns. Copies of the objects could be produced and then communicated to each application every time the state of the primary copy changes. However, this involves the performance cost of marshaling the objects. The problem is further compounded by the fact that marshaling capability would need to be explicitly implemented in either CORBA or COM. (Java RMI implicitly supports the capability to communicate objects by value.)

The fully-populated message approach is actually a stylized way of marshaling objects. While it is appealing to think of leveraging existing healthcare standards such as HL7, it is non-trivial to implement the parsers and translators to create and interpret these messages. Even if such an implementation was commercially available, it is not clear that it would be desirable to require that all of the applications in a shared context system be able to support HL7 messages.

The name-value pair approach represents the compromise that is pursued in this document. Using simple primitive data types enables the values of the items to be easily communicated between processes. Performance concerns are mitigated because an application will be able to examine the values of only those items of interest in a single out-of-process access. (The application simply indicates the names of the items whose values it is interested in.) The approach is also readily extensible, as new items (i.e., new name-value pairs) can easily be added to the set of items.

All of the context data representation approaches described above are subject to establishing semantic agreement about the meaning of the data. This is true whether the context data is represented as objects, messages, or name-value pairs. The process for establishing this agreement is beyond the scope of the CMA, and is instead specified in a series of HL7 context management subject-specific data definition documents. These data definitions are key to implementing a plug-and-play common clinical context system.

### 5.5 Context Data Access

Any common context architecture must provide a way for an application that has just started to obtain its initial view of the common context. The pull-model implicitly solves this problem. With the push-model, there are two basic approaches:

- When the application joins the common context system, the necessary data is pushed to it.
The data can be accessed from a well-known location, such as a file, or from the component that is responsible for pushing changes to the context system participants. This is, in effect, a specialized use of the pull-scheme.

The approach to this problem is linked to the approach by which applications access the context data for updating it, and the approach by which applications obtain the values for the context data when it has changed.

The options are straightforward:

- Each application maintains a copy of the context data. As changes occur, each application updates its local copy accordingly.
- A central “authentic” copy of the context data is maintained. Context data updates are directed by applications to this copy. Applications access this copy in order to inspect changes.

The approach in which each application maintains its own copy of the context data has an elegance to it. However, in the absence of an authentic copy, an application that has gotten out of synchrony with its peers may have a difficult time restoring its notion of the common context. Further, the communication costs of keeping all applications in synchrony can become significant, particularly as the complexity and size of the common context increases over time as additional common context items are defined.

The approach that is taken for the CMA is to maintain a single authentic copy of the common context for each common context system. Applications can choose to cache context data or they can simply access the authentic copy whenever they need to. Applications can also selectively read or write specific context data name-value pairs. Further, when the context changes, an application is only informed about the change and is not provided with the data that has changed. The application can selectively access this data when it needs to.

This approach was chosen as a balance between performance and complexity. Performance issues are addressed by enabling applications to have selective access to context data. Complexity issues are addressed by not forcing applications to maintain their own copy of the common context data.

5.6 Context Data Interpretation

In order for applications to apply common context data in a clinically consistent manner, they must interpret the meaning of the data in a uniform manner. With context items represented as name-value pairs, applications must be able to uniformly interpret both the meaning of the name and the value of a context item, or determine that it cannot correctly interpret the item.
Context data items are grouped by subject, wherein each subject contains a semantically related set of data items. There are two classes of subjects:

- An identity subject identifies a real-world entity (e.g., a specific patient) or concept (e.g., a specific encounter).
- An annotation subject contains data that describes or is otherwise pertinent to the entity or concept identified in an identity subject.

The context data items from which a subject may be comprised may represent the following categories of information:

- data that identifies a real-world entity or concept (such as a specific patient or a specific encounter),
- data that can be used to corroborate the identity data,
- data that annotates the identity data with additional information from an authentic source that describes the identified entity or concept.

Identity information is required in order to establish a common context between applications that involves a real-world entity or concept. Corroborating data can be used by applications and/or users as a basis for checking further that the identified entity or concept is what was expected. Annotation data can be used by applications as a means for interchanging data pertaining to the identified entity or concept, wherein this information is provided by an authentic data source designated by a site.

For example, a patient’s name can rarely be used to uniquely identify a patient. Typically, a medical record number or similar identifier that is generally unique over some population of patients for one or more clinical systems is used. However, these identifiers are rarely meaningful to the user. Corroborating data might be comprised of the patient’s name, sex, and data of birth. This data provides applications and/or the user with an additional means to check that the identified patient is the intended patient. Annotating data might be comprised of the patient’s marital status, home telephone number, and nationality. This data provides applications with a means to share data that describes a patient in addition to identifying the patient.

The clinical context is considered to have changed in a meaningful manner when identifier data is set. Applications are notified of changes to the context when identifier data, and possibly corroboration and/or annotation data, are set. Changes to corroboration data and/or annotation data that are not accompanied by associated changes to identifier data are not meaningful and are rejected.
5.6.1 Context Subject Data Definition Constraints

The specification for an identity subject shall include the data definition for at least one identifier data item and may include the data definitions for zero or more corroborating data items. Data definitions for annotation data items are not allowed for an identity subject.

The specification for an annotation subject shall include the data definition for at least one annotation data item. Data definitions for identifier and corroborating data items are not allowed for an annotation subject.

5.6.2 Establishing the Meaning of Context Data Item Names

Given this approach of organizing context data items into identifier, corroborating, and annotating data, there are two basic techniques for establishing the meaning of context item names:

- Apply a Context Management-specific information modeling process to identify and define candidate clinical context item names and meanings.

- Leverage names and their meaning as established by existing healthcare standards, such as the HL7 messaging standard.

The approach that is taken for the CMA is that existing HL7 messaging terms and their meaning are used as the default source for clinical context item names. New item names and associated meanings will be created only when the HL7 messaging standard is not applicable. The standard set of clinical context data context item names are specified in the document Health Level-Seven Standard Context Management Specification, Subject Data Definitions. Only the specified set of context data items shall be implemented by conformant systems.

The reason for this approach is that the value-added for HL7 context management is not in defining clinical content, but rather in enabling new forms of clinically-rooted desktop-based interoperability between independently-developed healthcare applications. There is little incentive to create new information models and develop new clinical concepts when there are existing concepts, such as those already specified for HL7 messaging, which can be leveraged.

5.6.3 Establishing the Meaning of Context Data Item Values

The abstract data types used to represent context data item values are also leveraged from the HL7 messaging standard. These types may be represented as strings encoded using a simple subset of the HL7 character encoding rules. These types may also be mapped into convenient technology-specific data types. The actual clinical context data context item data types are specified in the document Health Level-Seven Standard Context Management Specification, Subject Data Definitions.
There are two basic approaches for establishing the meaning of context item values:

- Assume that each item has a value that can be globally interpreted by all of the applications that share a common clinical context.
- Provide multiple values for each item name such that each value represents the same real-world entity or concept. Each application can apply the value it understands.

In some cases, it is safe to assume that a context item’s value can be globally interpreted by all applications. For example, if a patient’s date of birth is defined to be a corroborating context data item, the value of this item has a single global interpretation.

5.6.4 Representing Context Subjects That Cannot Be Uniquely Identified

Unfortunately, it is not possible to assume that all context subjects, such as patients, can be identified using globally unique identifier values. For example, a patient cannot necessarily be globally identified using a single identifier, such as a medical record number.

However, in these cases, there may be multiple synonymous identifier values, each of which is pertinent to a subset of the applications that share a common context. For example, a hospital and its affiliated clinics may assign their own medical record numbers to the same patient population. Applications, such as master patient index systems, enable tracking and mapping between these values. The result is multiple distinct values that identify the same patient.

It is not the purview of the CMA to resolve global identification issues. It is within the scope of the CMA to at least recognize that multiple identifier values may be necessary. Therefore, the approach taken in this document is to support multiple identifier values for context items when necessary.

An item that can have multiple values is actually represented as multiple items that have a common name prefix but use a distinct name suffix. The prefix for an item, and the constraints on values for the suffix, is defined in the document Health Level-Seven Standard Context Management Specification, Subject Data Definitions. The suffixes are configured into an application using an application-specific process when the application is installed at a site.

The values for such items are provided either by an application when it changes the clinical context, or by an external mapping agent. (See Chapter 5 Mapping Agents.)

5.6.5 Context Subjects and Item Name Format

All context items are organized by subject. Each subject represents a real-world entity or concept that is identified as part of the overall common clinical context. The name of a
context item includes the subject to which it belongs. The name of a context item also
indicates its role and meaning.

The scheme for assigning subject labels and item names must ensure that the complete name
for each item is unique across all item names for all subjects. To this end, standard subject
labels, an approach for assigning custom subject labels, and the format of the names of each
item that comprise the data for a subject, are defined in the document *Health Level-Seven
Standard Context Management Specification, Subject Data Definitions*.

### 5.6.6 Standard Context Subjects and Data Items

Each of the standard HL7 CMA subjects and associated context data items are defined in the
document *Health Level-Seven Standard Context Management Specification, Subject Data
Definitions*. This includes the core subjects of patient, encounter, observation, user, and
certificate, and their respective context data items.

### 5.6.7 Non-Standard Context Subjects and Data Items

Organizations, such as healthcare provider institutions and vendors, may define their own
context subjects and data items. These items are in addition to the standard subjects and the
standard items defined for the standard subjects. The scheme for defining custom subjects
and custom items for standard subjects is specified in the document *Health Level-Seven
Standard Context Management Specification, Subject Data Definitions*.

### 5.6.8 Representing “Null” Item Values

The value of a context data item can be set to the distinguished value of `null` to indicate that
the item does not have a valid value. This capability provides a means for an application to
explicitly indicate it has not set a valid value for a particular context item. For example,
setting the value of the identifier whose name is:

```
“Patient.Id.MRN.St_Elsewhere_Hospital”
```

to `null` indicates that the application has not set a valid value for this identifier.

The actual representation of `null` is technology-dependent and is specified in each of the HL7
context management technology mapping specification documents.

### 5.6.9 Representing an Empty Context Subject

A context identity subject is *empty* when a real-world entity or concept is not currently
identified. For example, for the patient subject, this means that a patient is not currently
identified.

An empty context subject is represented in either of two ways:
• There are no context identifier items.

• There are context identifier items, but the values for all of these items are null.

The initial state for all subjects in the context is that they do not contain any items. See Section 8.6 Context Change Transactions. An application can explicitly establish an empty context. See Section 8.10.3 Application Behavior with Regard to an Empty Context.

5.6.10 Case Sensitivity with Regard to Item Names and Item Values

Context item names are case insensitive. This means that case is not to be used for the purposes of comparing names. Further, the case used to represent the same item name can be different for different applications, and the case used to represent a particular item’s name at one time need not necessarily be the same at a later time. For example, the item names:

“Patient.Id.MRN.St_Elsewhere_Hospital”

“patient.id.mrn.st_elsewhere_hospital”

“PATIENT.ID.MRN.ST_ELSEWHERE_HOSPITAL”

are all equivalent.

A context item whose value is represented as a character string is also case insensitive, unless otherwise noted in the HL7 context management subject-specific data definition specification document that defines the item.

However, for consistency with the situations in which item values are case sensitive, the case used to represent the value for a particular item is preserved once the value has been set. The case for the item’s value is maintained until a different value is subsequently established for the item.

For example, the following flow of events is allowed:

1. An application sets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” to “RS779238XZW”.

2. An application gets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” as “RS779238XZW”.

3. An application sets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” to “AS119292RUH”.

4. An application gets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” as “AS119292RUH”.

Version CM-1.3 Copyright 2001, Health Level Seven
5. An application sets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” to “rs779238xzw”.

6. An application gets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” as “rs779238xzw”.

The following flow of events is not allowed:

7. An application sets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” to “RS779238XZW”.

8. An application gets the value of “Patient.Id.MRN.St_Elsewhere_Hospital” as “rs779238xzw”.
6 Component Model

The architecture for a common clinical context system is described in terms of components and the interfaces they must implement in order to be participants in the system. Only the components and interfaces that are germane to the establishment and maintenance of a common clinical context for a clinical desktop are described.

A role is described for each component, and the policies that govern the intended use of the interfaces are detailed. These policies can be thought of as the patterns of allowed interactions between components. Both normal and exceptional interactions are described.

The key components in a common clinical context system are a clinical context manager, one or more context participant applications, and an optional mapping agent for each context subject.

The context manager coordinates the applications each time there is a context change. It is also the “owner” of the authentic context for the system. The context participant applications set and/or get the context from the context manager. They must follow the policies established later in this document in order to behave as proper context management “citizens.”

A mapping agent is a service component that from the perspective of an application is a transparent participant in a context change. A mapping agent’s primary role is to add additional subject-specific context identifier items to the context data. This is useful when a subject is known to the various context participant applications via multiple distinct identifiers, but only one or a few of these identifiers are known to the application that sets the context.

Additional context management components are also defined, but serve in supporting roles. All of the necessary components are detailed later in this document.

The context manager does not need to know about the functionality or specific features implemented by any of the applications. Conversely, all applications perceive the context manager through a uniform set of interfaces and capabilities. Further, the applications do not need to know about each other in order to participate in the same context system. Finally, a mapping agent is transparent to applications, as it interacts only with the context manager.

Applications and the context management components can all be independently implemented and still interoperate as long as they comply with the CMA specification. The CMA specification is in turn predicated upon an underlying component model, described next.
6.1 Component and Interface Concepts

The clinical context manager and the applications that participate in a common context system are modeled in the architecture as components. The component model that is used is a high-level hybrid of the component models defined by Microsoft for its Component Object Model (COM) and by the Object Management Group for its Object Management Architecture (OMA).

6.1.1 Interfaces and References

In the hybrid model, components have one or more formally-defined object-oriented interfaces. Each interface defines a semantically related set of operations (methods) that the component is capable of performing. The interfaces implemented by a component represent the only way that other components can interact with it. Each interface is denoted by a reference that can be resolved at run-time to access the component instance that implements the interface.

Each method has a name and a set of inputs, outputs, and exceptions. The inputs enable a component’s clients to parameterize the behavior of the method each time they request that it be performed. The outputs enable the component to convey to a client the results that pertain to having properly performed the method. The exceptions enable the component to convey to a client the fact that something unexpected was encountered during the course of performing the method (such as an error condition). A method completes by returning outputs or by raising exceptions. Methods need not have inputs, outputs, or exceptions.

The methods defined for an interface are invoked using a binary calling sequence. This means that the component that issued the call does not need to be aware of how the component that services the call is implemented. The components might be implemented using different tools and libraries, and even different programming languages. Further, components can interact with each other in a location independent manner. A component only needs a reference to another component’s interface to perform calls against the component. Knowledge of the physical location of a component that services a call is not needed.

6.1.2 Interface Interrogation

The interfaces that a component implements can be determined by other components at run-time through direct interrogation. The interrogator uses the symbolic name of the interface, or an identifier that denotes the interface, to indicate the desired interface. If the interface exists, the component being interrogated returns a reference to the interface. Otherwise an error indication is returned.

It is assumed that all of the components defined in this document include a common method that enables interface interrogation. The name and signature for this method is the same for
all components implemented using a particular technology. The details of this method vary
for different implementation technologies and are not specified in this architecture document.

6.1.3 Principal Interface
Every component implements at least one well-known interface, referred to as the
component’s principal interface. The principal interface includes the same interface
interrogation method as a component’s other interfaces. The name of the principal interface
is the same for all components implemented using a particular technology. The principal
interface enables components to perform initial interface interrogations because the name of
the principal interface is known a priori, and because all components implement it.

The details of the principal interface and the methods that it supports vary for different
implementation technologies and are not specified in this architecture document.

6.1.4 Interface Reference Registry
An interface reference registry is a service that contains references to component interfaces.
Components can use the registry to obtain interface references to each other. A reference can
be used to access a component via the referenced interface. Each reference is denoted in the
registry by a symbolic name and/or description. This enables components to locate
references of interest based upon a symbolic and/or logical description of the reference of
interest.

It is assumed that an interface reference registry is provided by the underlying
implementation technology. The means by which interface references are denoted and placed
into the registry, and the means by which components access the registry to retrieve the
references, are technology-dependent.

The registry is assumed to be a well-known service that logically resides on each clinical
desktop. This means that each component on a desktop has an a priori technology-specific
means for knowing how to locate the desktop’s registry. This provides all components on a
desktop with a common means to obtain references to each other.

6.1.5 Interface Reference Management
To ensure orderly system behavior, components must have a means of knowing whether or
not other components possess references to any of its interfaces. This enables a component to
determine when it needs to be in a running state (because there is at least one other
component that possess a reference), and when it can terminate (because no components
possess a reference). The means by which this is accomplished is technology-specific.

It is assumed that each component that holds an interface reference performs an implicit or
explicit action, which is technology specific, that indicates it wants to use a particular
interface reference that it has obtained (e.g., from the interface reference registry). It is also assumed that a component performs an implicit or explicit action, which is technology-specific, when it no longer intends to use a particular reference. The latter action is referred to as disposing an interface reference.
7 General Theory of Operation for Clinical Links

Clinical links and the subjects that they represent are broadly classified as common or secure. The treatment of common links is described in Chapter 8 Patient Link, which is a common link for the patient subject, is detailed in Chapter 12 With a common link, any application can get or set the context data for the subject represented by the link.

The treatment of secure links is described in Chapter 11 User Link, which is a secure link for the user subject, is detailed in Chapter 13 With secure links, only applications with the appropriate access privileges are allowed by the context manager to set and/or get the context data for the subject represented by the link. Whether or not the subject represented by a link is secure is defined as part of the subject's data definition and is invariant across sites. However, for each subject that is defined as secure, the necessary application access privileges are configured on a site-specific basis.

There are, however, a number of concepts, rules, and policies that pertain to both common and secure clinical links in general, as described in this chapter.

7.1 Multiple Subjects and the Meaning of “Link”

Even though there are multiple subjects in a common context system (e.g., patient and user), there is only one link that coordinates the CMA-compliant applications on a desktop. This means that when an application is linked, it shall “tune” to all of the subjects it is capable of dealing with. For example:

- An application that is only Patient Link-enabled tunes to just the patient context.
- An application that is only User Link-enabled tunes to just the user context.
- An application that is both Patient Link-enabled and User Link-enabled tunes to both the patient context and the user context.

Conversely, when the user breaks an application’s link, then the application shall no longer be tuned to any context subject.

Independent of the number of context subjects it supports, a single visual cue shall be provided by an application to indicate whether or not it is linked. The appearance of this cue is defined in the each of the HL7 context management technology-specific user interface specification documents.

7.1.1 Context Manager Support for Multiple Context Subjects

The potential relationships between context subjects require that a context manager implementation have an understanding of multiple subjects and of the inter-relationships between subjects. Further,
applications will, in general, need to deal with multiple context subjects. There are two basic ways to address these issues:

- Maintain a context manager per subject, and enable context managers to interact with each other to maintain the overall context.
- Support multiple context subjects within a single context manager.

The first approach has the advantage that context manager implementations can be specialized to support a single subject. For example, this would enable a Patient Link context manager from one vendor to be used with a User Link context manager from another vendor. The disadvantages are that applications would need to deal with multiple context managers.

Further, the context managers would need some way to cooperate in order to coordinate transactions that affect multiple subjects. This coordination would require the definition of additional context manager interfaces. This coordination would also increase the complexity of the failure scenarios because of the increased opportunity for partial failures (e.g., one context manager fails while the other context manager continues to function).

The second approach has the advantage that it enables the complexities of dealing with multiple subjects to be hidden within the implementation of a single context manager. Additional context manager interfaces are not required, and partial failure scenarios are avoided. This approach also has the advantage that applications only need to deal with a single context manager.

The second approach has the disadvantage that context manager developers will need to support all subjects within their context managers. However, it is the CMA philosophy to push complexity into the context manager whenever it simplifies the creation of new applications and the reengineering of existing applications. The second approach is the one that is pursued because, from the perspective of an application, it is simpler than the first approach.

### 7.1.2 Effect of Multiple Subjects on Context Change Transaction

For application flexibility and backwards compatibility, it is highly desirable that:

- An application does not have to know about all possible subjects in order to set the context pertaining to just one subject.
- Either one, some, or all subjects can be set within a single context change transaction.

However, these desires raise the question of how to treat context data for a subject that is not “touched” during a transaction by the instigating application? There are two approaches:

1. At the completion of the transaction, an untouched subject is *empty*, meaning that it does not contain any context items.
2. At the completion of the transaction, the untouched subject is *unaffected*, meaning that it contains the same items and item values as it did before the transaction.

In the first approach, the context manager would ensure that each context change transaction begins with an empty context (i.e., all subjects are empty). With multiple subjects, only the subject that is touched during a transaction will contain items at the completion of the transaction.

However, a problem arises with this approach, as illustrated by the following example. An application that is only Patient Link-enabled might be co-resident with applications that are Patient Link and User Link-enabled. If the application that is only Patient Link-enabled changes the patient context, the user context shared by the other applications will be lost (i.e., it will be empty).

Applications could be required to know about all subjects and to explicitly copy the subjects that are not to be set from the current context to the new context. However, this creates a burden on the application developers. It is also a substantial impediment to backwards compatibility.

The second approach avoids this problem, as described in Section 7.1.3, *Context Manager Treatment of Multi-Subject Context Data*.

### 7.1.3 Context Manager Treatment of Multi-Subject Context Data

When a context change transaction is started, the context manager shall create a transaction-specific version of all of the context data. This version of the context data is initially empty and shall not contain any context items for any subject.

The application that instigated the transaction then establishes the new context by setting context data item values for one or more subjects. The application then informs the context manager that it has completed its context changes. The context manager shall then copy the items from the previous context to the new context for any subject that the instigating application did not touch. This shall occur before the context manager surveys the context participants.

The net effect is that the instigating application sets context items for whichever subject(s) it knows about. If a subject was “untouched” by the application, then the items for the subject are automatically post-filled by the context manager to reflect the values as they were *before* the context change transaction.

With these rules, an application can just set subjects based upon the user’s explicit gestures, such as selecting a patient, signing on, or both. An application only needs to set the subject context items that it is capable of setting. For example, an application may not be able to set all of the corroborating data for a subject. Similarly, a participant application does not have to deal with all subjects, or show all of the context data items defined for a subject.
7.1.4 Effect of Multiple Subjects on Mapping Agents

Each identity subject (e.g., patient) shall have at most one corresponding mapping agent. When a context change transaction reaches the phase during which the context manager instructs mapping agents to map the context data (i.e., context changes are pending), the context manager shall interact with each mapping agent in the order described in Chapter 9 Mapping Agents. Each mapping agent shall be informed only once per transaction that context changes are pending.

7.1.5 Effect of Multiple Subjects on Annotation Agents

Each annotation subject (e.g., certificate) shall have at most one corresponding annotation agent. When a context change transaction reaches the phase during which the context manager instructs annotation agents to set the context data (i.e., context changes are pending), the context manager shall interact with each annotation agent in the order described in Chapter 10 Annotation Agents. Each annotation agent shall be informed only once per transaction that context changes are pending.

7.1.6 Application Treatment of Multiple Subjects

An instigating application can set one, some, or all subjects in a single context change transaction. However, unless the user expects multiple subjects to change as a result of a gesture, it is recommended that an application generally set only one subject at a time. This enables the user to relate changes in the common context to gestures that they have explicitly performed. Cause-and-effect between a user’s gesture and a change in application state is an important element in creating systems that are easy for people to use.

An instigating application may also need to get the context data for various subjects at the conclusion of the transaction that it instigated. Specifically, if the application is interested in one or more annotation subjects, then the application will need to explicitly get the context items for these subjects. This is because the context item values for each annotation subject are set by a corresponding annotation agent during the course of the transaction but subsequent to when the instigating application completed its setting of subjects.

7.2 Context Subjects Dependencies

For simplicity, it is generally desirable that there not be any semantic dependencies between context subjects. This enables an application to set a context subject without concern for the other available subjects.

With this assumption, it is possible for an application to independently set the context data items for just one subject, some, or all subjects during the course of a single context change transaction. A context subject whose items have not been set by the application shall remain as it was prior to the transaction. (See Section 7.1.3 Context Manager Treatment of Multi-Subject Context Data.)
However, in certain cases there are real-world semantic dependencies between context subjects, wherein the context data item values for one subject must be consistent with the context data item values for one or more other subjects. An application that sets the context during a context change transaction needs to be cognizant of subject dependencies, however the context manager also plays a role in enforcing dependency rules.

The context manager cannot necessarily guarantee that dependent subjects have consistent context item data values. However, the context manager shall enforce rules that indicate application cognizance of the dependencies between the subjects whose context item data it sets.

These rules ensure that whenever a parent subject is set during a transaction, then each child subject that depends upon the parent subject is also set. Specifically, a child subject that is not set under these conditions shall be forced by the context manager to be empty at the conclusion of the transaction (i.e., the child subject’s pre-transaction value is not post-filled by the context manager, but rather the child subject is set to empty).

A child subject may always be set if its parent subject has not been set and if the parent’s pre-transaction value is not empty. If under these conditions a child subject is not set, then the child subject’s pre-transaction value shall be post-filled by the context manager.

If a parent subject is set to empty, or it is not set and its current value is empty, then each child subject may only be set to empty. This is the same thing that the context manager will automatically do under these conditions.

To illustrate the subject dependency rules enforced by the context manager, consider the example of the child subject Encounter, which is specified as being dependent upon the parent subject Patient.

If prior to a context change transaction both the Encounter and Patient subjects are already set and not empty, and during the transaction the application sets the Patient subject but not the Encounter subject, then at the end of the transaction the context manager shall ensure that Encounter subject is empty. This is in contrast to the context manager post-filling the context data items for the Encounter subject, as would be if the Encounter subject was not dependent upon the Patient subject. (See Section [7.1.3 Context Manager Treatment of Multi-Subject Context Data] for an explanation of post-filling context data items.)

Alternatively, if the context data items for the Encounter subject are set so that this subject is not empty, but the Patient context is empty, then this shall result in the context manager raising an exception to the application that instigated the transaction, and the context manager shall cancel the transaction.

In contrast, it is acceptable for an application to set the context data items for the Encounter subject even if the context items for the Patient subject are not set by the application during the transaction, as long as the Patient subject is not currently empty.
It shall be explicitly defined in each subject’s data definition as to whether or not the subject is
dependent upon one or more other subjects. A directed acyclic graph of dependent subjects is
allowed.

Whether or not a standard subject is dependent upon another subject is specified in the document
Health Level-Seven Standard Context Management Specification, Subject Data Definitions.

An identity subject may be dependent upon another identity subject. An identity subject shall never
be dependent upon an annotation subject. An annotation subject shall always be dependent upon an
identity subject. An annotation subject shall never be dependent upon another annotation subject.

Beholden to the rules above, subject dependencies for custom subjects, if any, may be defined by the
organization that specifies the custom subject. However, while a custom subject may be dependent
upon any other subject, a custom subject shall not require that a standard subject, or a custom subject
defined by another organization, be dependent upon it. For example, it cannot be asserted that
standard subject S is dependent upon custom subject C.

7.3 Empty Contexts Subjects

As described in Section 5.6.9 Representing an Empty Context Subject, applications can indicate that
a subject is empty. When a subject is explicitly set to empty, its context data is not post-filled by the
context manager.
8 Theory of Operation for Common Links

Common links involve context subjects for which application access privileges are not enforced by the context manager. Any application can get or set the context data for the subject represented by a common link. Common links form the foundation for CMA-based context management.

8.1 Component Architecture for Common Links

The following context management interfaces for common links are modeled and illustrated in Figure 11: Component Architecture

- **ContextManager** (CM) - implemented by the context manager; used by applications to join/leave a common context system and to indicate the start/end of a set of changes to the common context data.

- **ContextData** (CD) - implemented by the context manager; used by applications to set/get the data items that comprise the common context.

- **ContextParticipant** (CP) - implemented by an application that wants to participate in a common context system; used by the context manager to inform an application that the context has been set.

- **MappingAgent** (MA) - used by a context manager to inform a mapping agent that the clinical context has changes pending and that the mapping agent should perform its context data mapping responsibilities.

- **ContextAgent** (CA) - used by a context manager to inform a mapping agent or an annotation agent that the clinical context has changes pending and that the mapping agent or annotation should perform its context data mapping responsibilities. (Note that this is a new interface, and that mapping agent implementations may implement either MappingAgent or ContextAgent.)

- **ImplementationInformation** (II) – implemented by the context manager and mapping agents; used by applications, context management components, and tools, to obtain details about a component’s implementation, including its revision, when it was installed, etc.

Formal definitions of these interfaces, as well as example interactions between the components via these interfaces, are presented later in this document.
Figure 11: Component Architecture for Common Links
8.2 Mapping Agents

An optional mapping agent may also be part of the common context system. There may be at most one common mapping agent for each common subject. The context manager does not enforce access privileges for mapping agents that map common subjects (as opposed to secure subjects, for which access privileges are enforced).

For example, a patient mapping agent maps the identifiers for patients. Whenever an application sets the patient context, the context manager instructs the patient mapping agent (if present) to provide any additional identifiers it knows for the patient. The site-suffix for each of the mapped identifier items denotes the site for which the patient identifier is valid, for example:

<table>
<thead>
<tr>
<th>Example Item Names</th>
<th>Example Item Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient.Id.MRN.St_Elsewhere_Hospital</td>
<td>123-456-789Q36</td>
</tr>
<tr>
<td>Patient.Id.MRN.General_Hospital</td>
<td>6668-3923-987122</td>
</tr>
</tbody>
</table>

Mapping agents are described in more detail in Chapter 9.

8.3 Annotation Agents

An optional annotation agent may also be part of the common context system. There may be at most one annotation agent for each annotation subject. The context manager does not enforce access privileges for annotation agents that annotate common subjects (as opposed to secure annotation subjects, for which access privileges are enforced).

For example, a hypothetical demographics annotation agent would set an annotation subject that contains the demographic data for the current patient. Whenever an application sets the patient context, the context manager would instruct the demographics annotation agent (if present) to provide the data for the demographics subject, for example:

<table>
<thead>
<tr>
<th>Example Item Names</th>
<th>Example Item Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demographics.An.MothersMaidenName</td>
<td>Smith^^^^^</td>
</tr>
</tbody>
</table>

Annotation agents are described in more detail in Chapter 10.
**8.4 Context Change Transactions**

All changes to the common context are governed by a context change transaction that is initiated by an application but is coordinated by the context manager:

- An instigating application initiates a context change transaction and sets the context data within the context manager. This context contains the identity of the real-world entity or concept represented by each context subject.

- The context manager consults the mapping agents (if present) and they add data to the context manager’s context. This data includes additional identifiers by which the each real world entity or concept is known.

- The context manager consults the annotation agents (if present) and they add data to the context manager’s context. This data includes additional information that describes or is otherwise pertinent to each context subject.

- The context manager surveys the other applications, and if the transaction completes, they obtain the context data from the context manager.

As an example, the high-level events that transpire when a user selects a patient are summarized in Figure 12. This description assumes that a patient mapping agent and a hypothetical demographics annotation agent are present. The patient mapping agent is presumed to know the identifiers for all patients for all applications within the common context system. The demographics annotation agent is presumed to know the demographics for all patients within the common context system.
The details for how this process works and the responsibilities of the applications and CMA
components are described next.

8.5 Joining the Common Context System

Applications join a common context system via the context manager for the system. The context
manager’s ContextManager interface is used for this purpose. The application obtains a reference to
this interface by interrogating the context manager’s principal interface. A reference to the context
manager’s principal interface is obtained from the desktop’s interface reference registry.

An application typically retrieves the current common context data from the context manager’s
ContextData interface in order to establish its initial context. A reference to the context manager’s
ContextData interface is obtained by interrogating the context manager’s principal interface or by
interrogating the context manager’s ContextManager interface. The context data is represented as a
set of name-value pair items.

8.6 Context Change Transactions

Once it is a participant within a common context system, the context manager will inform the
application of context data changes through the application’s ContextParticipant interface. This data
can be set by any of the participants in the common context system. A participant executes a context
change transaction to effect a context change. The transaction is coordinated by the context manager
and involves the instigator of the transaction as well as the other participants.

The ContextManager interface is for beginning and ending a context change transaction. The
ContextData interface is used for setting the new context data.

When a context change transaction is started, the context manager creates a transaction-specific
version of the context data. This version of the context data is initially empty and does not contain
any name-value pair items. This is to prevent data from the current context from becoming mixed
with the data for the new context. Items are added to the transaction-specific context data during the
course of the transaction.

This version of the context data is updated during the course of the transaction and is intended to be
visible only to the application that instigated the transaction. All other applications continue to view
the context data as it was when most recently published. The published context data is replaced with
the context data set during the course of the transaction when the transaction completes successfully.

Prior to the first context change transaction, the published set of context data items is empty. Items
are added during the course of subsequent transactions.

While the context manager serves as a holder for the current context data, its semantic understanding
of the meaning of this data is intended to be minimal. Further, the specific items that constitute the
context data are not assumed to be hardwired into the context manager implementation. This enables new context items to be defined over time without requiring changes to context manager implementations. This includes context items that represent identifier data as well as corroboration data.

Only one context change transaction is allowed at a time. Once it has started a change transaction, the instigator of the transaction is free to update the context data via the context manager’s ContextData and/or SecureContextData interfaces.

### 8.7 Transactional Consistency

In order to ensure that changes to this set of items are self-consistent, a participant must explicitly begin and end a context data change transaction. All of the context change operations that are performed within the scope of the transaction are treated as a single logical unit of work. When the transaction completes, either all of the changes are published, or none of them are. Other participants that access the ContextData and/or SecureContextData interfaces to read the context data values will see the values as they were prior to the transaction. Only the instigator of the transaction will see the values as they are during the course of the transaction. This prevents other participants from accidentally seeing inconsistent values.

This capability relies upon the proper use of context coupons, which are monotonically increasing identifiers that are assigned each time a change transaction begins. The context manager provides the instigator of a transaction with the context coupon when it is started. All other participants can only obtain from the context manager the coupon for the most recently committed transaction. A coupon is also provided as a parameter to most of the methods defined for the ContextData interface, thereby enabling the manager to determine whether it should respond in terms of the transaction-in-progress or the most recently committed transaction.

When the instigator of the context changes is done, it informs the context manager that the changes have been completed. A context manager may unilaterally decide to terminate a transaction and undo the changes if an application fails to indicate that it is done with its changes in a timely manner. (The context manager decides how long “timely” is. How this value is determined is an implementation decision.)

### 8.8 Context Change Notification Process

When the instigator completes the context changes, the context manager initiates a two-step change notification process wherein it determines whether to publish the shared context data changes. This process is inspired by the two-phase commit protocol used in many database systems to ensure transaction consistency. For the purposes of managing a common clinical context, the protocol has been simplified.
In the first step of the process, the context manager surveys the applications. Each application is informed that there are a candidate set of context data changes and is asked to indicate whether it can accept these changes. At this point, applications are provided with the context coupon value for this change transaction. This enables the applications to access the context data changes in order to consider specific data values as part of their decision about whether to accept the changes. This is accomplished via the context manager’s ContextData and/or SecureContextData interfaces. It is possible for a participant to obtain just the values for the subjects that have been set.

The context manager gathers the results of the survey and provides them to the application that instigated the context change. Depending upon the survey responses the application may be free to go ahead and publish the changes, or it may need to solicit guidance from the user about how to proceed. This guidance is required when there is at least one surveyed application that:

- is unable to apply the context change because it is blocked (e.g., it is a single threaded application that has a modal dialog open); these applications are referred to as “busy”

- might lose work performed by the user if it applies the context changes (e.g., the user was in the process of entering data that would not be applicable in the new context); these applications are referred to as having “conditionally accepted” the context changes.

For each application in one of these states, the user is provided with a description that identifies the application and explains its situation.

When user guidance is required, the following choices are offered:

- **Cancel** - the context change is canceled; the context changes are not published.

- **Break Link** - the context changes are applied just to the application with which the user initiated the context changes. This application essentially breaks away from the common context system until the user explicitly instructs the application to rejoin the system. The application that has broken away displays a distinct visual cue indicating that its context may be different from the other applications (e.g., it might display a warning message in a prominent location).

- **Apply** - the context data changes are applied to all of the applications, including those that indicated that they might lose work performed by the user; *this choice is allowed only when there are no busy applications.*

It is the responsibility of any application that enables the user to instigate a context change to present, when necessary, a dialog that obtains the user’s guidance as described above. The appearance of the dialog and the commands that the user can choose from are specified in each of the HL7 context

---

2 A specific visual cue will be recommended within each of the HL7 context management technology-specific user interface specification documents.
management technology-specific user interface specification documents. This will ensure a consistent and familiar set of interactions for users across CMA-conformant applications.

The ability for any one application to require the user’s direct involvement in mediating context changes provides an important efficiency and safety feature.

The efficiency feature addresses the fact that changing the context may cause an application to lose work performed by the user. This work may be in the form of data entered but not yet saved by the user, or may be in the form of an expensive computation (such as a lengthy database retrieval) that would need to be re-performed in light of a context change. Allowing the user to decide how to proceed in these circumstances minimizes the likelihood that the user will unintentionally lose work.

The safety feature addresses the fact that it may not always be possible to force an application to accept changes to the context data. Specifically, this is the case for blocked, or busy, applications.

If context changes were automatically applied piecemeal to just the applications that could respond, applications could become out of synchrony with regard to their clinical context, without the user being aware of the situation. For example, the user might assume that after a context change, all of the applications are displaying data for the same patient when in fact they are displaying data for different patients. The approach described above avoids this problem. This is because the only time that an application can become out of synchrony with regard to the clinical context used by the other applications is when the user has explicitly instructed it to break away.

In the second step of the two-step change notification process, the applications in the common context system are informed about whether or not the context changes are to be applied. If all of the surveyed applications indicate that they accept the changes, then the changes are applied and are reflected as the new context state. If the user indicates that the changes should be canceled, then the changes are discarded.

Once a participant has been informed that the context data has been set, it is free to inspect the data to obtain the new values if it has not already done so (again, using the context manager’s ContextData and/or SecureContextData interfaces). The participants can also assume that all of the other participants are applying the same context data.

Subsequent to the completion of the context change transaction, the application that instigated the context change transaction may also want to inspect the context data. This is necessary if the application is interested in annotation data that might have been set by one or more annotation agent during the transaction.

In either case, the context change transaction completes when all of the applications have been informed of the outcome of the survey. If the context manager is unable to inform an application of the survey outcome, it will keep trying periodically, unless the manager determines that the
application has terminated. The periodic attempt to notify a non-responsive application does not prevent the transaction from completing, nor will it prevent a new transaction from being started.

### 8.9 Leaving a Common Context System

When an application terminates, it explicitly leaves the common context system by informing the context manager via its ContextManager interface. At this time, the context manager shall dispose of any application interface references that it possesses, and the application shall dispose of any context manager interface references that it possesses.

A diagram of the overall common context system model is presented in [Figure 13](#) followed by component interaction diagrams that represent typical common context data update transactions.

### 8.10 Behavioral Details

#### 8.10.1 Application Behavior When it Cannot Cancel Context Changes

It is possible that an application that instigated a context change transaction cannot easily implement the capability to cancel the transaction. In this case, it is acceptable for the application to not offer canceling the transaction as an option to the user. The details of how this appears to the user are specified in each of the HL7 context management technology-specific user interface specification documents.

#### 8.10.2 Application Behavior When it Does Not Understand Context Identifiers

It is possible that an application is unable to interpret any of the context identifier items that were set when the current context was established by another application. For example, the selected patient might not be a patient known to the application.

An application that is unable to interpret any of the identifiers shall still participate in the context change transaction. This situation is not a basis for the application to prevent the transaction from proceeding. Specifically, the application shall not use the surveying process to reject the context change.

However, at the completion of the transaction, the application shall clearly indicate to the user that it is unable to apply the current context. The application shall not show any patient data. The details of how this indication appears to the user are specified in each of the HL7 context management technology-specific user interface specification documents.
8.10.3 Application Behavior with Regard to an Empty Context

The context is empty when a context system is first initialized. (See Section 5.6.9, Representing an Empty Context Subject). When this is the case, all of the applications in the context system shall clearly indicate to the user that there is no current context. The details of how this indication appears to the user are specified in each of the HL7 context management technology-specific user interface specification documents.

8.10.4 Surveying Details

During the context change survey, the context manager informs each of the applications in the common context system (except for the application that instigated the changes) that there are pending context data changes. When an application is surveyed, it shall create a visual cue that indicates it is about to change its clinical context before responding to the survey. It shall not change its context yet. The context-changes-pending indication shall only be removed once the context manager has informed the surveyed application about how to proceed.

Under normal circumstances, the application will eventually be notified by the context manager about whether or not the context changes should be applied. However, if the context manager is unable to inform the application about how to proceed (e.g., because the application blocked after responding to the survey but before being notified that the context changes have been accepted), the user will at least be able to determine that the application may not be in synchrony with the other applications. This is because the application is presumably still displaying a visual cue that indicates it might change its clinical context. The fact that this cue is still being displayed after the context has been set clues the user that there is a problem with the application.

An application can explicitly respond to a context change notification survey by indicating one of the following:

- **Accept**: It is willing to accept the context data changes and to change its internal state accordingly if the changes are published.

- **Accept-Conditional**: It is in the midst of a task that might cause work to be lost if the user does not complete the task; if the changes are published it is willing to terminate the task, accept the context data changes and change its internal state accordingly.

If the changes are subsequently published, an application can defer changing its internal state until some time in the future (for example, when it regains the focus for user-inputs). However, it must

---

3 A specific visual cue recommended within each of the HL7 context management technology-specific user interface specification documents.
offer a visual cue that indicates it is not in synchrony with the new context. For example, it might blank out its data display or minimize itself.

An application that cannot interpret the context data (e.g., does not know who the patient is) shall accept the changes. However, the application shall clearly indicate to the user (e.g., by displaying a message) that it cannot apply the current context data.

The context manager infers an implicit response from an application under the following conditions:

- **Terminated**: the context manager has determined that the application has terminated without first informing the context manager

- **Busy**: the context manager has determined that the application is still running but is unable to answer the survey (e.g., the application is single-threaded and has a modal dialog open)

It is not possible for a surveyed application to explicitly reject, and therefore prevent, a context change.

The context manager gathers the survey responses and returns them to the application that was used to instigate the context change transaction. Applications that have responded with `accept-conditional` are expected to also provide a succinct but informative description of the consequences to the user of applying the context changes. The context manager then prepends the name of the application (provided by the application when it joined the common context system) to the description. This description is shown to the user by the instigating application.

The context manager also provides the instigating application with a succinct but informative description about any applications that are busy. This description includes the name of the application. This information is provided by the context manager on behalf of these applications, as they are unable to do so for themselves. This description is also shown to the user by the instigating application.

Applications that have terminated do not affect the survey process. The context manager considers such applications to no longer be part of the common context system. Any information that the manager is maintaining about terminated applications is discarded.

Applications that have suspended their participation in the context are not involved in the survey process.

Applications that have joined the system but indicated that they do not want to participate in surveys are not involved in the survey. However, they are informed along with the other participants.

---

4 A specific visual cue is recommended within each of the HL7 context management technology-specific user interface specification documents.
whenever the decision to accept the changes is published. (They are not informed about decisions to cancel changes, as this information would be irrelevant.)

### 8.11 Common Clinical Context Use Model

The Common Clinical Context Use Model (Figure 13) illustrates a system with four actors (Authorized User, Healthcare Application, Context Manager, and a System’s Administrator) applying forces on three use cases. The use cases are Lifecycle of Common Context, Context Selection Change, and Abnormal Termination of Common Context.

---

**Figure 13: Common Clinical Context Use Model**

The common clinical context system is presented by providing a diagram of each use case followed by interaction diagrams illustrating different behavioral flows of the associated use case. Each use case has an associated description, which is provided below. Further, for brevity the specific interface names (ContextManager, ContextParticipant, and ContextData) are not used; their abbreviations are used instead (CM, CP, and CD). Also, the word “interface” is abbreviated to “iface”. The diagram notes (illustrated as a sheet of paper with corner folded over) are from a software developer’s perspective, not the user of the application.

#### 8.11.1 Lifecycle of Common Context

A common context does not initially exist. An application must establish the common context. The common context ceases to exist when there are no longer any applications participating in the common context. [Figure 14] Interaction Diagram 1, and Interaction Diagram 2 illustrate this use case.
Figure 14: Common Context Lifecycle Use Case
Interaction Diagram 1: Common Context Lifecycle

User

Application AAA

ContextManager

I choose "Jane Doe"

CM::JoinCommonContext(CP iface of AAA, surveyYes)

CM::StartContextChanges

context coupon

CD::SetItemValues

CM::EndContextChanges

survey results empty

CM::PublishChangesDecision("accept")

Possibly more transactions.

Exit program

CM::LeaveCommonContext

Did last participant leave?

Yes

Exit

MostRecentContext coupon=0
No items in the context.

Transaction Begins

Single participant, therefore, no survey is required.

Transaction Complete

MostRecentContext coupon has a unique value.
Items with values now in the context.
Interaction Diagram 2: Suspending/Resuming Context Participation
8.11.2 Context Selection Change Use Case

The Context Selection Change use case assumes a patient context has been established. The user is currently focused on one application, while several other healthcare applications may be executing on the same host machine. The user chooses to change the selected patient from “Jane Doe” to “Sam Smith”.

Figure 15 illustrates this use case. There are several possible instances of this use case which are provided in Interaction Diagram 3 through Interaction Diagram 10.

Figure 15: Context Selection Change Use Case
Context Management Specification, Technology and Subject-Independent Component Architecture

I choose "Sam Smith"

CM::StartContextChanges

CD::SetItemValues

CM::EndContextChanges

Possibly concurrent surveys.

survey results: all applications accept

CM::PublishChangesDecision("accept")

Possibly concurrent notifications.

"Sam Smith"

Possibly concurrent queries.

CP::ContextChangesPending

"accept"

Possibly concurrent surveys.

CP::ContextChangesPending

"accept"

Possibly concurrent queries.

CD::SetItemValues

CD::SetItemValues

Interaction Diagram 3: All applications accept the changes
Interaction Diagram 4: An application conditionally accepts the changes; user decides to cancel changes
Interaction Diagram 5: An application does not respond to survey

User is told that Application CCC did not respond to pending changes survey.

Application CCC busy and does not respond.

Application CCC not notified of cancellation.
Interaction Diagram 6: An application does not respond to change notification

I choose "Sam Smith"

CM::StartContextChanges

CD::SetItemValues

CM::EndContextChanges

Survey results: all applications accept

CM::PublishChangesDecision("accept")

Application CCC, busy and does not respond.

Context Manager responsible for attempting to notify until a new transaction begins.

Same interaction would occur if pending changes were canceled.
Interaction Diagram 7: An application responds after context change transaction has completed
Interaction Diagram 8: A non-surveyed application participates in context change

User

Application AAA

I choose "Sam Smith"

Application Manager

Application BBB

Application CCC

I choose "Sam Smith"

CD::GetItemValues

CM::JoinCommonContext(CP iface of AAA, surveyYes)

CM::JoinCommonContext(CP iface of BBB, surveyYes)

CM::JoinCommonContext(CP iface of CCC, surveyNo)

CM::StartContextChanges

CM::EndContextChanges

CD::SetItemValues

CM::PublishChangesDecision("accept")

CP::ContextChangesAccepted

CP::ContextChangesPending

"accept"

Application CCC notified of context change.

Application does not want to be involved with context change surveys.

Application CCC not surveyed.

CM::JoinCommonContext(CP iface of AAA, surveyYes)

CM::JoinCommonContext(CP iface of BBB, surveyYes)

CM::JoinCommonContext(CP iface of CCC, surveyNo)

CD::GetItemValues

CD::GetItemValues

Copyright 2001, Health Level Seven
User is told that Application CCC might lose work in progress. User is provided with a description of consequences of context change.

"Are you sure you want to change?"

User decides to accept consequences of change.

"I choose "Sam Smith""

Survey results: Application CCC conditionally accepted with this reason.

CM::StartContextChanges
CD::SetItemValues
CM::EndContextChanges

CP::ContextChangesAccepted
CP::ContextChangesPending
"accept"
"conditionally accept" and consequences

Provides description of consequences of context change being accepted.

Interaction Diagram 9: An application conditionally accepts the changes; user decides to accept consequences of change
User is told that Application CCC might lose work in progress. User is provided with a description of consequences of a context change at this time.

"Are you sure you want to change?"

I choose "Sam Smith"

User is told that Application CCC might lose work in progress. User is provided with a description of consequences of a context change at this time.

"Are you sure you want to change?"

I choose "Sam Smith"

Context Management Specification, Technology and Subject-Independent Component Architecture

Interaction Diagram 10: An application conditionally accepts the changes; user breaks link with common context

Copyright 2001, Health Level Seven

Version CM-1.3
8.11.3 Abnormal Termination of Common Context Use Case

The Abnormal Termination of Common Context Use Case involves a system administrator forcing the termination of the context manager through some action. The common context participants are notified of the termination of the common context.

Figure 16 illustrates the abnormal termination use case while Interaction Diagram 11 captures an instance of this case.

Figure 16: Abnormal Termination of Common Context Use Case
Interaction Diagram 11: Abnormal Termination of Common Context
8.12 Optimizations

There are several optimizations that have been designed into the specification. These optimizations are reflected in the interface specifications described in Chapter 15:

- An application can indicate that it never wants to participate in the survey conducted by the context manager when the context data changes. The context manager will assume that such applications always accept the changes. Read-only data displays represent a class of applications for which this capability is useful.

- An application can selectively suspend its participation in the surveying process without actually leaving the common context. This enables an application to perform computational tasks without being interrupted by context changes. This also enables an application to minimize its use of computational resources if it is in a state (e.g., minimized) in which responding to context changes provides no benefit to the user. The application can subsequently resume its participation in the common context. The capability to suspend and resume is an optimized alternative to joining and leaving the common context.

- An application can obtain just the context items from the subjects that were altered by the most recent change transaction. This capability will become increasingly useful as additional common context data items are defined.

- Multiple common context items can be accessed by an application in a single invocation of a context manager method. This optimizes performance by reducing the number of calls an application needs to make to access context items.

- When an application is notified about a context change, it is also provided with the context coupon value that it needs in order to access the context data. This simplifies the design of applications because they do not necessarily need to keep track of context coupon values.

- Context managers can be implemented to conduct the change survey and the subsequent change notifications in a concurrent manner, thereby decreasing the amount of time it takes to complete these computations.

Additional optimizations, such as enabling applications to indicate their interest in only being notified when specific context data items change are candidates for future enhancements.
8.13 The Simplest Application

The responsibilities that an application must implement in order to behave properly as a participant in a common context system depends upon the application’s functionality. Applications that need to participate in the context change survey must implement straightforward but non-trivial behaviors. However, for many applications it will suffice to implement a very small set of behaviors. Specifically, the simplest participants are those that do not participate in the survey, do not set the context data, and only want to be informed when context changes have been accepted. These applications only need to do the following:

1. Join the common context system via the context manager’s ContextManager interface.
2. Implement the ContextParticipant method that enables the application to be informed about accepted context changes.
3. Access the context data via the context manager’s ContextData interface.
4. Leave the common context system upon termination, via the context manager’s ContextManager interface.

As Interaction Diagram 12 illustrates below, this amounts to implementing one method for ContextParticipant. (The others can be stubbed with trivial default behaviors.) It also requires using two ContextManager methods: one to join and one to leave a common context system. Finally, it requires using one ContextData method to access the context data. The application does not necessarily need to keep track of the value of the context change coupon, as the context manager each time a change occurs provides the correct coupon value to the notified application. The result is that simple applications are not penalized for being co-participants with applications that have more sophisticated needs.
I choose "Sam Smith"

"Sam Smith"

survey results: all applications accept

CM::JoinCommonContext(0P iface to AAA, surveyYes)
CM::StartContextChanges
CD::SetItemValues
CM::EndContextChanges

CM::PublishChangesDecision("accept")

CM::JoinCommonContext(0P iface to BBB, surveyNo)
CP::ContextChangesAccepted
CD::GetItemValues
item values
CM::LeaveCommonContext

Interaction Diagram 12: Simplest Application
9 Mapping Agents

A mapping agent is a context agent that provides a common context system with a means to automatically supply multiple synonymous identifiers for the same real-world entity or concept, even when only one identifier is known to the application used to instigate a context change. This mapping is performed in a manner that is transparent to the user and to the applications in the context system.

For example, multiple medical record numbers within a healthcare enterprise might identify a patient. However, each application might only be able to denote a particular patient via just one of these identifiers. When the user selects a patient using such an application, the application sets the new patient context using the patient identifier it knows. The context manager automatically delegates the task of mapping the provided identifier to additional identifiers to a mapping agent. A master patient index system might serve as the basis for implementing a mapping agent capable of mapping patient identifiers.

Mapping agents are not necessarily needed in order to realize a useful and correctly functioning common context system. Specifically, mapping agents are not needed when each real-world entity or concept has a single identifier that is already known to all of the applications in the common context system. For example, there are healthcare enterprises that have a uniform way to identify their patients. A patient mapping agent for a context system in such an enterprise is not necessary.

9.1 Assumptions and Assertions

It is not an objective of the CMA to define how mapping agents should be implemented or to prescribe or assume a particular mapping agent implementation. Instead, a mapping agent is treated as an abstraction. Interfaces are defined that enable mapping agents to be connected to context managers for the purpose of aiding in the mapping of context identifiers between multiple identifier spaces.

Additional assumptions and assertions include:

- When present, the mapping agent is the authority within a common context system on the mapping between context identifiers for a specific identity subject.

- There may be at most one mapping agent per subject, and the agent for a subject may only map the data for that subject.

- A mapping agent does not allow an identifier to map to more than one real-world entity or concept (e.g., a patient mapping agent does not allow a patient identifier to map to more than one patient).
• There is at most one mapping agent per context subject per context system. (Behind
  the “scenes” mapping agents may work together, or may be implemented using a
  single common service. However, this is not visible to the context manager or the
  context participants.)

• A context manager does not know about the mapping agent implementation; a
  context manager only “sees” a mapping agent through its CMA-defined interface.

• Context participant applications do not “know” about the mapping agent (or even if
  there is one); the mapping agent does not “know” about context participant
  applications.

• The mapping agent may reside on a computer that is remote from the computer (s)
  upon which the context manager(s) they serve reside; however, these computers must
  be connected by a LAN or WAN whose performance is LAN-equivalent.

• Mapping agents are an optional component of a CMA context management system.

9.2 Interfaces

The following interfaces are defined for and implemented by mapping agents:

• MappingAgent (MA) - used by a context manager to inform a mapping agent that the
  clinical context has changes pending and that the mapping agent should perform its
  context data mapping responsibilities.

• ContextAgent (CA) – an optimized variant of the interface MappingAgent.

• ImplementationInformation (II) - used by a context manager to obtain details about
  who implemented the mapping agent, when it was installed, etc., for the purpose of
  creating detailed error reports.

In addition, mapping agents set/get context data items using the context manager’s
ContextData and/or SecureContextData interfaces.

The mapping agent interfaces are modeled and illustrated in Figure 11: Component
Architecture

9.3 Theory of Operation

Assume, first, that one or more context participants have already joined the same common
context and that they are connected to the context manager. Further, assume that the context
manager already has an interface reference to each mapping agent’s MappingAgent (or
ContextAgent) interface. How these references are obtained is described in Section 9.3.2, Initializing a Context System When a Mapping Agent is Present.

Given these conditions, a context participant instigates a context change transaction via the context manager’s ContextManager interface, sets the new context data via context manager’s ContextData or SecureContextData interface, and then indicates it is done setting the data via the context manager’s ContextManager interface.

At this point, before the other context participants are surveyed, the manager informs each mapping agent that the context data has changes pending, via each mapping agent’s MappingAgent interface, or if implemented, ContextAgent interface. The proposed context data items that are available to the mapping agent are exactly as the instigating participant set them.

If the ContextAgent interface is used, then the context manager presents the proposed context data to be mapped directly to the mapping agent. Specifically, the data for the context subject that the mapping agent is supposed to map is presented to the agent. If the MappingAgent interface is used, then the mapping agent needs to read the appropriate proposed context data via the context manager’s ContextData interface. In either case, the mapping agent may set one or more additional context data identifier items, and may also set corroborating data items, via the ContextData interface.

A mapping agent is only allowed to map data for the context subject that it is the mapping agent for (i.e., the patient mapping agent may only map the context data for the patient subject). A mapping agent may only add data to the subject. If a mapping agent attempts to map data for another subject, or to set the value for a context data item whose value has already been set, then the context manager shall raise an exception.

Applications (including the instigating application) are not allowed to set context item values after the instigating application has completed its changes. However, the context manager allows the mapping agents to make changes because it knows it is a mapping agent that is setting the item values. How the context manager knows that it is a mapping agent is described later.

Once the mapping agent has completed its mapping tasks, the context manager surveys the context participants and processing of the context change transaction is performed as usual. With this approach, all of the synonymous values for an identifier will be set before the other applications are informed via a context manager-initiated survey that the context has been set.

However, if the instigating application has set multiple values for a context identifier, and the mapping agent detects an inconsistency among these values, then it informs the context manager that the context change transaction has been invalidated. This is because the mapping agent is the authority in a context system when it comes to mappings between...
identifiers. Allowing the transaction to proceed could create confusion about the context among the other context participants.

The details about the conditions under which a mapping agent can invalidate a context change transaction are described in 9.3.7 Conditions for Mapping Agent Invalidation of Context Changes.

When the mapping agent invalidates a context change transaction, the context manager does not survey the participating applications. Instead, the context manager informs the instigating application that the transaction has been invalidated. The instigating application then asks the user to intervene to decide how to proceed.

The user can decide (via a dialog presented by the application that was used to instigate the context change) whether to cancel the context change or to break the instigating application away from the common context system. In either case, the context change transaction is terminated and the context changes are discarded. Additional identifiers are not mapped and the other applications are not surveyed.

This approach gives the user the option of applying the context changes to just the application used to instigate the context change while also preventing the other applications from becoming confused about the context.

The details of this situation are described in 9.3.8 Treatment of Mapping Agent Invalidation of Context Changes.

9.3.1 Mapping Agents and Secure Context Subjects

Only the mapping agent designated for setting the context for a specific secure context subject shall be allowed by a context manager to set the data for the subject. A mapping agent for a secure context subject must implement all or the policies and behaviors specified for See Chapter 11, Theory of Operation for Secure Links.

9.3.2 Initializing a Context System When a Mapping Agent is Present

A mapping agent and the context manager it serves must be connected to each other. There are two ways in which this can be accomplished. Either the context manager connects to the mapping agent, or the mapping agent connects to the context manager. The order in which this connection occurs has significant impact on complexity and computing resource utilization.

The mapping agent could conceivably locate and connect to a context manager the same way a context participant does. This requires that the connection be made before the first time a context participant application sets the context. This is so that the mapping agent can be instructed by the context manager to perform its mapping tasks.
A consequence of this approach is that a context manager will execute even if it is not actively servicing any context participants. Further, the requirement that the connection be made before the first time a context participant application sets the context introduces initialization-sequeencing complexities.

In general there is no way to know when the first context participant will connect to a context manager, so the only prudent recourse would be to launch the context manager and the mapping agent as part of the boot-up process for the desktop they serve. This would complicate the installation process for context managers and mapping agents.

The alternative is for the context manager to connect to the mapping agent. This approach enables the connection to be deferred until the mapping agent is needed to service a context participant. However, a means by which context managers can locate the necessary mapping agent must be established.

Fortunately, the fact that there is only one mapping agent per context subject per clinical desktop enables the location process to be easily implemented using the desktop’s technology-specific desktop interface reference registry. Specifically, a reference to a mapping agent’s principal interface is entered into the desktop’s interface reference registry. The symbolic name and/or description of the interface within the registry indicates the context subject that the mapping agent maps. The context manager obtains this reference and uses it to interrogate the mapping agent to obtain references to its other interfaces, such as MappingAgent or ContextAgent.

An additional benefit of the manager-connects-with-agent approach is that it is not even necessary for distinct connect/disconnect methods to be defined. Instead, the context manager simply informs the mapping agent whenever the context manager has changes pending. The context manager explicitly provides a reference to its principal interface to the mapping agent. The mapping agent then interrogates the context manager via its principal interface to obtain a reference to other context manager agent interfaces, such as the interface ContextData.

The sequence of events is shown in Interaction Diagram 13: Context Change Transaction with Mapping Agent.

### 9.3.3 Calling Sequence When Multiple Mapping Agents are Present

The order in which mapping agents for unrelated context subjects are informed by the context manager about a pending context change is not specified. The context manager may inform unrelated mapping agents about a context change in a serial or concurrent manner.

However, a mapping agent for a dependent subject is informed about a pending context change only after the mapping agent for the subject it depends on has been informed. The
order in which mapping agents for context subjects that depend upon the same subject are
informed is not specified.

9.3.4 Terminating a Context System When a Mapping Agent is Present

To enable the orderly termination of the context system, the context manager shall implicitly
or explicitly dispose of any mapping agent interface references that it possesses prior to
terminating. The mapping agent shall dispose of any context manager interface references
that it possesses when it has completed its mapping actions for a context change transaction.
The means by which these disposals are effected is technology-specific.

The consequence of these disposals is that at the end of a context change transaction, only
context participant applications will possess context manager interface references. If there
are no participants, then the context manager can properly terminate. (Participants dispose of
any context manager interface references that they possess prior to terminating. See Section
6.1.5 Interface Reference Management.) This also means that once the context manager
terminates, the mapping agent can also properly terminate.
9.3.5 Distinguishing Between Mapping Agents and Context Participants

When a mapping agent is informed that a context change is pending, the context manager provides it with two coupons. One coupon denotes the context change transaction; the other denotes the mapping agent. The mapping agent coupon is not the same as any of the coupons assigned by the context manager to the context participants. The mapping agent shall use the coupon that denotes it whenever it sets context data via the ContextData interface. The context manager uses this coupon to determine that a mapping agent, and not a context participant, is setting the context data. Only context agents, which

Interaction Diagram 13: Context Change Transaction with Mapping Agent

- Context Manager
- Context Participant
- Mapping Agent

- ContextManager::JoinCommonContext()
- ContextManager::StartContextChanges()
- ContextData::SetItemValues()
- ContextManager::EndContextChanges()
- MappingAgent::ContextChangesPending(Principal iface to context manager)
- Mapping agent locates context manager’s ContextData interface
- ContextData::GetItemValues()
- ContextData::SetItemValues()
- Return from ContextChangesPending
- Surveying the other context participants occurs here
- Return from EndContextChanges()
includes mapping agents, are allowed to set context data after the instigator of the context change has indicated that it has completed the context changes.

9.3.6 Mapping Agent Updates to Context Data

A mapping agent shall only add additional context identifier and corroborating data items to the context as part of its mapping activities. Additions to the context for these types of context data items are primarily for the benefit of the context participants other than the application that instigated the context change. This is because it cannot be assumed that the instigating application will re-read these context data items once it has completed its context changes. In contrast, the other applications do not read the new context until they are surveyed, which occurs after the mapping agent has added data to the context.

If a mapping agent was allowed to change the values for identifier or corroborating context data items that have been set by the instigating application, it could be confusing to the user. This is because the user might see differences between the context data as displayed by the instigating application and as displayed by the other context participant applications.

Given this concern, a mapping agent shall not alter the values of any of the identifier or corroborating context data items that have already been set by the instigating participant as part of the proposed context. Any attempt to alter existing context data items by the mapping agent shall result in the context manager raising an exception.

A mapping agent shall not delete identifier or corroborating context data items. Any attempt to delete context data items by the mapping agent shall result in the context manager raising an exception.

9.3.7 Conditions for Mapping Agent Invalidation of Context Changes

A context subject is comprised of a set of context data items, each of which is represented as name/value pairs (see Section 5.4, Context Data Representation, and Section 5.6, Context Data Interpretation). It is the responsibility of every application that sets these items to ensure that they are self-consistent. However, there are a variety of potential item name and/or item value inconsistencies that a mapping agent must be able to detect.

Specifically, if an application has set multiple values for a context identifier item, and the mapping agent determines that these values do not all identify the same real-world entity or concept (e.g., patient), the mapping agent shall invalidate the context change transaction.

Specifically, a mapping agent shall invalidate a context change transaction when:

- The instigating application sets more than one value for the same context identifier item, but the mapping agent determines that at least two of these values identify different entities or concepts.
The instigating application sets more than one value for the same context identifier item, but the mapping agent knows that at least one of these values is inconsistent with the other values.

There are situations in which the mapping agent must not invalidate a context change transaction even though there are apparent context item inconsistencies. A mapping agent must not flag what it believes to be inconsistencies when in fact the suspect items might represent reasonable application behaviors.

The following scenarios illustrate the desired mapping agent behaviors, in this case for a patient mapping agent. Assume that there are two patients, each with identifiers for two sites, and the mapping agent is able to map the patient identifiers for both sites:

<table>
<thead>
<tr>
<th>Patients and Their Site-Specific Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Institution</td>
</tr>
<tr>
<td>St. Elsewhere Hospital</td>
</tr>
<tr>
<td>St. Elsewhere Clinic</td>
</tr>
</tbody>
</table>

The first two scenarios represent inconsistencies that the mapping agent must respond by invalidating the context change transaction. The last three scenarios represent inconsistencies that the mapping agent must ignore:
<table>
<thead>
<tr>
<th>What the instigating application does …</th>
<th>Example …</th>
<th>What the mapping agent does …</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Sets two identifier values, both with the intent of denoting John Doe, but the values erroneously denote John Doe and Jim Smith.</td>
<td><em>Item identifies John Doe:</em> `[Patient.Id.MRN.St_Elsewhere_Hospital, 123-456-789Q36] <em>Item erroneously identifies Jim Smith:</em> [Patient.Id.MRN.St_Elsewhere_Clinic, 18291-81293-D812]</td>
<td><strong>Invalidates</strong> the context change transaction because the first identifier value denotes John Doe, while the second denotes Jim Smith. Mapping is not performed.</td>
</tr>
<tr>
<td>2 Sets more than one identifier pair, both with the intent of denoting John Doe. The first value is John Doe’s hospital identifier, but the second value is not John Doe’s clinic identifier.</td>
<td><em>Item identifies John Doe:</em> [Patient.Id.MRN.St_Elsewhere_Hospital, 123-456-789Q36] <em>Item does not identify John Doe:</em> [Patient.Id.MRN.St_Elsewhere_Clinic, 0000-00000-0000]</td>
<td><strong>Invalidates</strong> the context change transaction because while the first identifier value is John Doe’s hospital identifier, the second value is known not to be John Doe’s clinic identifier. Mapping is not performed.</td>
</tr>
<tr>
<td>3 Sets only one context identifier item and the name of the item is not known to the mapping agent.</td>
<td><em>Item name not known to mapping agent:</em> [Patient.Id.MRN.General_Hospital, 6668-3923-987122]</td>
<td><strong>Ignores</strong> this situation and does not inform the context manager about inconsistencies. Mapping is not performed.</td>
</tr>
<tr>
<td>4 Sets more than one value for a context identifier item, and one or more of the item names are not known to the mapping agent.</td>
<td><em>Item name known to mapping agent:</em> [Patient.Id.MRN.St_Elsewhere_Hospital, 123-456-789Q36] <em>Item name not known to mapping agent:</em> [Patient.Id.MRN.General_Hospital, 6668-3923-987122]</td>
<td><strong>Ignores</strong> this situation and does not inform the context manager about inconsistencies. Mapping is performed.</td>
</tr>
<tr>
<td>5 Sets the corroborating or annotating data to values that are different (or incomplete) as compared to the corroborating or annotating data known to the mapping agent.</td>
<td>Application sets corroborating data containing the identified patient’s name to “Jack Doe” but mapping agent knows the identified patient as “John Doe”.</td>
<td><strong>Ignores</strong> this situation and does not inform the context manager about inconsistencies. Mapping is performed.</td>
</tr>
</tbody>
</table>

1 In summary, detectable inconsistencies between identifier values are the only reason that a mapping agent shall invalidate a transaction. Transactions shall not be invalidated when unknown identifier names are used by an application or because of corroborating or annotating data inconsistencies.

9.3.8 Treatment of Mapping Agent Invalidation of Context Changes

Applications that instigate context change transactions and then explicitly set more than one identifier during a context change transaction shall explicitly handle the situation in which a mapping agent invalidates a context change transaction. (Applications that set only one identifier do not need to handle this situation.)
An instigating application is not provided with a means to distinguish between the
invalidation of a context change transaction and the presence of a busy application. These are
clearly different situations, but are to be handled by an instigating application in the same
way. The application shall present a dialog that clearly indicates that a problem has been
encountered while attempting to change the common context.

The dialog shall include a description of the problem that was encountered. The dialog shall
also enable the user to cancel the context change or to break the link between the instigating
applications and the other applications.

When the mapping agent has invalidated a transaction it shall not be possible for the user to
force a common context change. If the user decides to break the link between the instigating
application and the other applications, instigating application shall only apply the context
change to itself. This application shall break away from the common context and shall
clearly indicate to the user that it is not participating in the common context.

If the user cancels the context change, then the instigating application shall indicate this fact
to the context manager. Both the instigating application and the context manager shall
discard the current transaction. The context manager shall not survey the other applications.

Independent of the reason for which the mapping agent invalidated the transaction, the
context manager shall always provide to the instigating application the same user-friendly
description of the problem that was encountered. This is in order to keep things simple for
the user, who is unlikely to be concerned about the details of what went wrong. This
description shall be included in the dialog by the instigating application.

The appearance of the dialog and the commands that the user can choose from are specified
in each of the HL7 context management technology-specific user interface specification
documents. The wording for the user-friendly description that is included in the dialog is also
specified in these documents. This will ensure a consistent and familiar set of interactions for
users across CMA-conformant applications.

The sequence of events that occur when a mapping agent invalidates a context change
transaction is shown in Interaction Diagram 14: Mapping Agent Invalidates Context Change
Transaction.

9.3.9 Mapping Null-Valued Identifiers

A mapping agent shall not perform any mapping when the context subject is empty (See
Section 5.6.9 Representing an Empty Context). The net effect is that the context subject
remains empty, and all of the applications see the context as such.
9.3.10 Initializing Mapping Agents

Different mapping agent implementations may require different initialization methods. For example, a mapping agent might need to authenticate the current user in order to enforce security policies. Other than being automatically launched by a context manager, the additional steps needed to initialize a mapping agent are implementation issues and are not
addressed by this specification. (Future versions of the CMA specification may provide
standardized ways of initializing mapping agents.)

It can be the case that different mapping agent implementations will require different explicit
or implicit actions on the part of the user to complete their initialization tasks. An example of
an explicit user action is signing on to the mapping agent via a mapping agent-supplied
dialog. An example of an implicit user action is signing on to a context participant
application that relays its authentication of the user to the mapping agent; this obviously
implies a relationship with the mapping agent that goes beyond this specification.

9.3.11 Handling Mapping Agent Failures

A context manager must be able to detect and handle the failure of a mapping agent.
Specifically, a context manager shall behave in a robust manner even if its calls to a mapping
agent’s MappingAgent interface do not return in a timely manner.

The recourse, after a timeout has occurred, is for the context manager to continue with the
normal processing of the context change transaction. If the mapping agent has indeed failed,
then some of the context participants may not be able to interpret the next context. However,
this fail-soft approach still enables the user to perform useful work until the mapping agent
failure is corrected.

Finally, even if a mapping agent has failed, a context manager shall continue to try to access
the mapping agent during subsequent transactions on the prospect that the failure has been
corrected. In doing so, the context manager may need to obtain a new interface reference for
the mapping agent (because the old reference may no longer be valid).

Note that this policy of continually attempting to access a failed mapping agent also applies
even when a context manager is first launched. It may be the case that a mapping agent
becomes available after the context manager has begun executing. (See Section 9.3.10
Initializing Mapping Agents for one explanation of why this might happen.) A context
manager that does not locate and initiate a mapping agent when it is launched shall
nevertheless keep trying between and/or during context change transactions. It is an
implementation decision as to how the performance impact of this policy is minimized.

9.4 Mapping Agent Effect on Application Security Policies

Mapping agents may implement their own security policies in terms of what context data it
will map for a particular user. Mapping agent security policies can differ from the policies of
the participating applications. For example, a mapping agent’s policies might effect what
patients a user can, or cannot, access.
When the mapping agent’s policy is more restrictive than one or more of the participating application’s, a mapping agent might elect to not map an identifier because doing so would violate the security rules known to the mapping agent. When the mapping agent’s policy is less restrictive than one or more of the participating applications, each application’s own security policy will be the predominating policy for the current change transaction.

A mapping agent that elects to not map an identifier because of security concerns shall not indicate this fact to the user. The user will simply observe that access to the selected subject is not possible through one or more of the participating applications. These applications do not know that the identifier for the selected patient has not been mapped because of the mapping agent’s security policy. Instead, it looks to the applications as though a subject has been selected but the identifier(s) by which the subject is known to the applications has not been provided. These applications behave as specified for in Section 8.10.2 Application Behavior When it Does Not Understand Context Identifiers.

9.5 Identifying Mapping Agent Implementations

Context managers use a mapping agent’s ImplementationInformation interface to provide system administrators with a description of the mapping agent implementation it is using. This information can help system administrators diagnose run-time problems that involve mapping agents.

The ImplementationInformation interface shall be supported by all mapping agent implementations. A context manager shall not interact with a mapping agent that does not support this interface.

9.6 Performance Costs and Optimizations

When present, a mapping agent will be involved in every context change transaction. This adds an overhead to the context change transaction in the form of the added communication between the context manager and the mapping agent, and for the time it takes for the mapping agent to validate the identifiers and provide any additional mappings for the identifiers. However, these costs are viewed as being worth the benefits of the semantic integrity that a mapping agent brings to a context system.

In some cases, a mapping agent will be implemented using an underlying application that provides its own user interface, for example, for patient selection. This type of mapping agent is, in effect, both a mapping agent and a context participant application. In the case in which this underlying application is used to instigate a context change, performing identifier validations and mappings is superfluous. It is possible to optimize the mapping agent implementation so that it does not perform identifier validations and mappings when it knows that it was essentially itself that instigated a context change.
However, the only information that is readily available to the mapping agent that could help it determine this fact is the context change coupon. This coupon is provided by the context manager to an application when the application starts a context change transaction. This coupon is also provided by the context manager to the mapping agent via its MappingAgent interface, or its ContextAgent interface if implemented, during each context change transaction.

It is an implementation decision as to how the portion of an application that implements a mapping agent obtains the value of the context coupon from the portion of the application that instigates a context change transaction.
10 Annotation Agents

An annotation agent is a context agent that provides a common context system with a means to automatically supply additional data that describes a particular context subject.

This data is provided in a manner that is transparent to the user and to the applications in the context system.

For example, a user certificate annotation agent might provide the data that represents a users digital certificate. This certificate would be the certificate for the user currently identified in the user subject.

In some cases an annotation agent for a specific annotation subject may be an optional enhancement to a context system. In other cases, in order to achieve the desired context sharing capabilities, an annotation agent for a specific annotation subject may be a required component.

10.1 Assumptions and Assertions

It is not an objective of the CMA to define how annotation agents should be implemented or to prescribe or assume a particular annotation agent implementation. Instead, an annotation agent is treated as an abstraction. Interfaces are defined that enable annotation agents to be connected to context managers for the purpose of setting annotation data during the course of a context change transaction.

Additional assumptions and assertions include:

• When present, the annotation agent is the authority within a common context system on the content for its annotation subject.

• There is at most one annotation agent per annotation subject per context system. (Behind the “scenes” annotation agents may work together, or may be implemented using a single common service. However, this is not visible to the context manager or the context participants.)

• A context manager does not know about the annotation agent implementation; a context manager only “sees” an annotation agent through its CMA-defined interface.

• Context participant applications do not “know” about the annotation agent (or even if there is one); the annotation agent does not “know” about context participant applications.
• The annotation agent may reside on a computer that is remote from the computer(s) upon which the context manager(s) they serve reside; however, these computers must be connected by a LAN or WAN whose performance is LAN-equivalent.

• Annotation agents are an optional component of a CMA context management system.

10.2 Interfaces

The following interfaces are defined for and implemented by mapping agents:

• ContextAgent (CA) - used by a context manager to inform an annotation agent that the clinical context has changes pending and that the annotation agent should set the necessary annotation data.

• ImplementationInformation (II) - used by a context manager to obtain details about who implemented the annotation agent, when it was installed, etc., for the purpose of creating detailed error reports.

In addition, annotation agents set/get context data items using the context manager’s ContextData and/or SecureContextData interfaces.

The annotation agent interfaces are modeled and illustrated in Figure 11: Component Architecture.

10.3 Theory of Operation

The theory of operation for an annotation agent is essentially the same as for a mapping agent. (See Chapter 9, Mapping Agents.) However:

• Annotation agents are not informed about the pending context change until all of the mapping agents have been informed and have completed their mapping process.

• An annotation agent may not set data for a context subject other than the annotation subject that it corresponds to. An annotation agent that attempts to do so will be presented with an exception by the context manager.

• Only the annotation agent may set the context data for the context subject that it is the agent for. Other agents, and context participants, may not set the data for the agent’s subject.

• An annotation agent may not invalidate a context change transaction.
Further, annotation subjects can be secure. Only the annotation agent designated for setting the context for a specific secure annotation subject shall be allowed by a context manager to set the data for the subject. See Chapter 11, Theory of Operation for Secure Links.
11 Theory of Operation for Secure Links

This chapter describes Context Management Architecture (CMA) support for secure links. With secure links, only applications with the appropriate access privileges are allowed by the context manager to set and/or get the context data for the subject represented by the link. Whether or not the context subject represented by a link is secure is defined in the subject’s data definition and is invariant across sites. However, the necessary application access privileges are configured on a site-specific basis.

Secure links are an extension of CMA support for common links. Specifically, support for secure links add security capabilities that not only enable the creation of the User Link capability, but which also serve as a foundation for future subjects that require security.

In order to accomplish this, the architecture for common links is extended. The interfaces defined for common links are used unchanged: ContextManager, ContextParticipant, MappingAgent, and ImplementationInformation. Two additional security-related interfaces are defined: SecureContextData, which is modeled upon the common link interface ContextData, and SecureBinding, which enables a trusted relationship to be established between applications and components.

Additional capabilities for secure links include:

- The provider institution decides which applications are to be trusted.
- In keeping with the CMA philosophy, the approach is conceived for low re-engineering costs.

The architecture that supports these capabilities is described next.

11.1 Secure Link Terms

The following terms are used to describe the theory of operation for secure links:

- **Secure Link-enabled application** - an application that implements the CMA secure link capability.

- **Empty context** – a context is not defined for a particular identity subject, either because no context identifier items are present in the subject’s context data (as is the case when a context manager is first initialized) or because the values of all of the identifier items for the subject that are present in the context data are null (as is the case when an application explicitly indicates that the context is empty).
11.2 **Desktop Assumptions**

The following assumptions are made about the clinical desktop upon which Secure Link-enabled applications are deployed:

- The desktops upon which Secure Link-enabled applications are deployed may reside in physically unsecured locations.

- While recommended, it may not be the case that appropriate security precautions have been taken to restrict the types of operating system-level actions, such as installing new programs, that users can perform on desktops that reside in physically unsecured locations.

In summary, the CMA is intended to be no less secure than the Secure Link-enabled applications would be were they not Secure Link-enabled. In general, Secure Link-enabled applications will be substantially more secure.

11.3 **Secure Link Common Context System Description**

Consistent with the CMA support for common links, on each desktop there are applications that are context participants, and there is a context manager. The applications perform context change transactions.

However, the secure context is communicated throughout the common context system in a secure manner. This is to prevent people from accidentally or maliciously controlling applications that are Secure Link-enabled.

The necessary security is achieved by adding capabilities to the CMA that enable the realization of a “chain of trust” among the Secure Link-enabled applications and associated CMA components components. With the chain of trust, Secure Link-enabled applications and CMA components work together to mutually authenticate their mutual interactions, thereby providing the basis for ensuring that only authorized users are allowed access to a common context system.

The chain of trust not only simplifies the overall solution, but results in a system that is more secure than would be the case if authentication data were part of the common context, and were therefore vulnerable to security attacks directed against the context manager or context agent.

The chain of trust is specified in Chapter 14.
11.3.1 Authenticating Secure Subject Access

The data definition for each context subject shall stipulate whether or not the subject is secure. A secure subject is one for which attempts by an application or agent to set and/or get the subject’s data must first be authenticated by the context manager. In so doing, the context manager must ensure that the application or agent is among those listed as being allowed to set and/or get the subject’s data. The list of applications and agent access privileges is configured into the context manager on a site-specific basis. The means for doing this are not specified by the CMA.

There are two types of security available for secure subjects. Subjects for which applications and agents must have appropriate privileges to set and/or get the subject’s data shall be specified in their data definitions as “Secure subject, authenticated sets and gets.” Subjects for which applications and agents must have appropriate privileges to set the subject’s data, but for which any application or agent may get the subject’s data, shall be specified in their data definitions as “Secure subject, authenticated sets only.”

11.3.2 Secure Mapping Agent

A secure mapping agent may be part of the common context system. A secure mapping agent implements the CMA security policies and secure interfaces defined for implementing secure links. There may be at most one secure mapping agent for each secure subject. The context manager enforces access privileges for mapping agents that map secure subjects.

11.3.3 Secure Annotation Agent

A secure annotation agent may be part of the common context system. A secure annotation agent implements the CMA security policies and secure interfaces defined for implementing secure links. There may be at most one secure annotation agent for each secure annotation subject. The context manager enforces access privileges for annotation agents that set secure subjects.

11.3.4 Context Management Interfaces

The context management interfaces defined for secure links are similar to the ones defined for common links. A context participant still implements ContextParticipant (CP). The context manager still implements ContextManager (CM), but it also implements the following new interfaces:

- SecureContextData (SD) - Similar to the ContextData interface defined for common links, this interface is used by applications to securely set/get the values for the items (logically represented as name-value pairs) that comprise the clinical context.

- SecureBinding (SB) - Used by applications to establish a secure communications binding with the context manager before using the SecureContextData interface.
The interfaces implemented by a secure mapping agent are MappingAgent (MA) and ImplementationInformation (II). These are the same interfaces as defined for a common mapping agent.

11.3.5 Overall Secure Link Component Architecture

The overall Secure Link architecture (including the Common Link Architecture) is illustrated in Figure 17: Component Architecture for Secure Links. (A description for how to interpret the notation used in this diagram appears in the appendix Diagramming Conventions.)

![Figure 17: Component Architecture for Secure Links](image)

<table>
<thead>
<tr>
<th>Component Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA = ContextAgent</td>
</tr>
<tr>
<td>CD = ContextData</td>
</tr>
<tr>
<td>CM = ContextManager</td>
</tr>
<tr>
<td>CP = ContextParticipant</td>
</tr>
<tr>
<td>MA = MappingAgent</td>
</tr>
<tr>
<td>SB = SecureBinding</td>
</tr>
<tr>
<td>SD = SecureContextData</td>
</tr>
</tbody>
</table>

Note – A context agent that is a mapping agent may implement MA instead of CA. However, CA enables more efficient agent implementations.
11.4 Process for Setting a Secure Context Subject

The process for performing a context change transaction to set a secure context is essentially the same as defined for common subjects:

- An instigating application initiates a context change transaction and sets the secure context within the context manager. This context contains the appropriate data for the secure subject.

- The context manager consults the secure mapping agent (if present) and it adds data to the context manager’s secure context.

- The context manager surveys the other applications, and if the transaction completes, they obtain pertinent secure context data from the context manager.

As an example using “User Link”, the high-level events that transpire when an application sets secure context data are summarized in Figure 20: User Link Context Change Process. This description assumes that a user mapping agent is present. The user mapping agent is presumed to know the logon names for all users for all applications. (See Section 13.6.1 [User Mapping Agent]). The description omits most of the details pertaining to the surveying of the participant applications by the context manager. This process is identical to the process defined for Patient Link. (See Chapter 8.)

Figure 18: User Link Sign-On Process
11.5 Designating Applications for Setting/Getting Secure Subjects

Any Secure Link-enabled application has the potential to set secure context data for use by all other Secure Link-enabled applications on a desktop. Similarly, any Secure Link-enabled application has the potential to get secure context data as it has been set by another Secure Link-enabled application on a desktop.

However, depending upon the type of security defined for a particular context subject, each site may designate on a per-application basis whether or not the Secure Link-enabled application is allowed to get and/or set the context data items for the subject. Only the applications designated as being allowed to set the data items for the subject shall be allowed by a context manager to do so. Only the applications designated as being allowed to get the data items for the subject shall be allowed by a context manager to do so.

The one exception to this rule is that any application shall be allowed to set any secure subject to empty.

A context manager implementation-specific configuration process is used to designate, for each secure subject, which applications are allowed to set and/or get the items for the subject. One, several, or all of the Secure Link-enabled applications on a desktop can be designated as being allowed to set and/or get the data items for a secure subject. It is recommend that a healthcare institution analyze the use cases for their clinical applications to determine how to best deploy Secure Link-enabled applications.

The decision criteria for a provider institution’s choice of whether to designate an application for setting and/or getting secure subject data is based upon whether they trust the application’s security capabilities as it pertains to that secure subject data. For example, for applications that set the user context (i.e. User Link), it might not be a good choice to designate an application that maintains user passwords in plain text (which can easily be read by unauthorized users).

11.6 Busy Applications

When a context change transaction is conducted, it is possible that an application is unable to participate because it is busy. For example, a single-threaded application that has a modal dialog open will not be able to respond until the dialog is closed.

This situation is dealt with for secure links in the same way as for common links. Specifically, a busy application effectively prevents a context change transaction from occurring. The only option for the application that instigated the transaction is to ask the user if they want to break the link.
However, breaking the link has the potential to compromise security. As an example using User Link, with a broken link, multiple users could effectively be logged on to different applications on the same desktop.

This situation is not substantially different from breaking the link for a common subject, which results in different applications on the same desktop being tuned to different common subjects.
12 Patient Link Theory of Operation

Patient Link enables the user to select a patient once, from any Patient Link-enabled application, as the means for automatically “tuning” all of the Patient Link-enabled applications in the common context system to the same patient. Patient Link is an exemplar of a common link.

12.1 Patient Link Component Architecture

Patient Link is a common link. The optional additional component of a patient mapping agent is defined, as described below.

12.2 Patient Subject

The context identity subject of Patient is defined for Patient Link. The context data identifier item for this subject is an alphanumeric patient identifier, such as a medical record number. The patient’s name is not used as an identifier.

The Patient subject is not dependent upon any other subject.

This identifier is unlikely to be universally unique. However, it is assumed that a population of patients across which the identifier is unique can be established. Each such population is referred to as a site, as it is typical that each population of patients corresponds to a physical site within an overall healthcare institution.

Consequently, a single patient may be identified using multiple patient subject identifier items. Each item is differentiated by a different site-specific suffix. An application shall be configurable such that it can be instructed on-site as to which suffix (of suffices) it is to use when it interacts with the context manager to set or get patient context data.

The format of a patient subject identifier item name includes a site-specific suffix. Use of this suffix, and the values that may be assigned to this suffix, is at the discretion of each healthcare institution at which a context management system is deployed.

In addition to identifier items, the patient subject also supports corroborating and annotating data items. The actual names, meaning, and data types used to represent the values for these context data items are defined in the document Health Level-Seven Standard Subject Data Definitions.
An example of a patient subject identifier item appears below:

<table>
<thead>
<tr>
<th>Example Item Name</th>
<th>Example Item Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient.Id.MRN.St_Elsewhere_Hospital</td>
<td>RAS1958-12939213-122</td>
</tr>
</tbody>
</table>

12.3 **Patient Mapping Agent**

An optional patient mapping agent is also part of the common context system. The patient mapping agent maps the identifiers for patients. Whenever an application sets the patient context, the context manager instructs the patient mapping agent (if present) to provide any additional identifiers it knows for the patient. The site-suffix for each of the mapped identifier items denotes the site for which the patient identifier is valid, for example:

<table>
<thead>
<tr>
<th>Example Item Names</th>
<th>Example Item Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient.Id.MRN.St_Elsewhere_Hospital</td>
<td>123-456-789Q36</td>
</tr>
<tr>
<td>Patient.Id.MRN.General_Hospital</td>
<td>6668-3923-987122</td>
</tr>
</tbody>
</table>

Mapping agents are described in detail in Chapter [ ]

12.4 **Patient Link Context Change Process**

The process for performing a context change transaction is as defined for common links in general. (See Section [8.4](#) Context Change Transactions)

12.5 **Stat Admissions**

A stat admission occurs when an application needs to enable the user to record information about a patient even if an identifier for the patient is not known. In this case, the application should indicate to the user that it is breaking its participation in the patient context, and then break its participation upon user confirmation. This is because it is not possible for the application to identify the patient, which is needed in order to change the common context. The only reasonable recourse is for the application to break its participation in the common context.
13 User Link Theory of Operation

This chapter describes CMA support for User Link. With User Link, a user can securely sign on to any User Link-enabled application on a desktop using just one logon name and one means of authentication (such as a password) in order to securely sign on to all User Link-enabled applications on the desktop.

User Link is an exemplar of a secure link. It also adds the following additional capabilities:

- The provider institution decides which applications are to be trusted to authenticate users.
- There can be multiple ways to authenticate users, including passwords, biometrics, etc.
- In keeping with the CMA philosophy, the User Link approach is conceived for low re-engineering costs.

The architecture that supports these capabilities is described next.

13.1 User Link Terms

The following terms are used to describe the User Link theory of operation:

- **User Link-enabled application** - an application that implements the CMA User Link capability.
- **Sign on** – the act of identifying oneself to an application, prior to initiating a user session, in a manner that can be authenticated by the application, typically involving a secret password or a biometric reading (such as a thumb-print scan).
- **Log-off** – the termination of a user’s session with an application.

13.2 Desktop Assumptions

The following assumptions are made about the clinical desktop upon which User Link-enabled applications are deployed:

- Logging-off from an application does not require user authentication.
- The desktops upon which User Link-enabled applications are deployed may reside in physically unsecured locations.
• While recommended, it may not be the case that appropriate security precautions have been taken to restrict the types of operating system-level actions, such as installing new programs, that users can perform on desktops that reside in physically unsecured locations.

In summary, the CMA is intended to be no less secure than the User Linked applications would be were they not User Linked. In general, User Linked applications will be substantially more secure.

13.3 User Link Component Architecture

User Link is a secure link. The architecture for User Link also defines a user mapping agent and an authentication repository as optional additional components, as described below.

13.4 User Subject

The context identity subject of User is defined for User Link. This is a secure subject, with authenticated sets only. The context data identifier item for this subject is the user’s logon name. A logon name denotes a user to an application. A user’s logon name is generally different from their given name.

The User subject is not dependent upon any other subject.

This identifier is unlikely to be universally unique. However, it is assumed that a population of users across which each logon name is unique can be established. Each such population is referred to as an application, as it is typical that within an overall healthcare institution each population of users corresponds to a particular application.

Consequently, a single user may be identified using multiple user subject identifier items. Each item is differentiated by a different application-specific suffix. An application shall be configurable such that it can be instructed on-site as to which suffix (or suffices) it is to use when it interacts with the context manager to set or get user context data.

The format of a user subject identifier item name includes an application-specific suffix. Use of this suffix, and the values that may be assigned to this suffix, is at the discretion of each healthcare institution at which a context management system is deployed.

In addition to identifier items, the user subject also supports corroborating data items. The actual names, meaning, and data types used to represent these context data items are defined in the document Health Level-Seven Standard Context Management Specification, Data Definition: User Subject.
An example of a user subject identifier item appears below:

<table>
<thead>
<tr>
<th>Example Item Name:</th>
<th>Example Item Value:</th>
</tr>
</thead>
<tbody>
<tr>
<td>User.Id.Logon.3M_Clinical_Workstation</td>
<td>robs</td>
</tr>
</tbody>
</table>

13.5 **User Authentication Data Is Not Part of the User Context**

The data used to authenticate a user is *not* included as part of the user context data. This data is typically a password, but it can be any data that is used to authenticate a user, such as a biometric sample. Instead, each application is expected to be able to sign on a user given just the application-specific logon name for the user.

This approach substantially reduces security risks because the data used by an application to authenticate the user remains private to the application. If this data were part of the user context, it would be vulnerable to undesired access. However, in order for applications to tune to the user context, they must trust that the context data is authentic. The means by which this is accomplished is referred to as the “chain of trust” and is described below.

13.6 **User Link Common Context System Description**

A common context system that supports User Link includes context participant applications, a context manager, and two optional components: a user mapping agent, and an authentication repository. These optional components are described next.

13.6.1 **User Mapping Agent**

An optional user mapping agent may be part of the common context system that supports User Link. The user mapping agent maps the logon names for users. The user mapping agent is similar to, but distinct from, the patient mapping agent (although a single mapping agent implementation could fulfill both roles).

Whenever an application sets the user context, the context manager instructs the user mapping agent (if present) to provide any additional logon names it knows for the user. The application suffix for each of the mapped identifier items denotes the application for which the mapped logon name is valid, for example:

<table>
<thead>
<tr>
<th>Example Item Names:</th>
<th>Example Item Values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>User.Id.Logon.3M_Clinical_Workstation</td>
<td>robs</td>
</tr>
<tr>
<td>User.Id.Logon.Medicalogic_Logician</td>
<td>rob_seliger</td>
</tr>
<tr>
<td>User.Id.Logon.HP_CareVue</td>
<td>r_seliger</td>
</tr>
</tbody>
</table>
13.6.2 Authentication Repository

In order to make it practical to re-engineer existing applications to support the chain of trust, the CMA authentication repository component is defined. This repository enables applications to securely store and retrieve application-specific user authentication data. The repository is used by applications that do not have a built-in means to easily sign on a user given only a logon name.

The authentication repository implements the following interfaces:

- AuthenticationRepository (AR) - Used by applications to securely interact with the repository to store and retrieve user authentication data.
- SecureBinding (SB) – Used by applications to establish a secure communications binding with the repository before using the AuthenticationRepository interface. This is the same interface that the context manager implements.
- ImplementationInformation (II) – Originally defined for the patient mapping agent, this interface is added to the authentication repository so that applications, other components, and tools, can obtain details about the authentication repository, including its revision, when it was installed, etc.

13.6.3 Overall User Link Component Architecture

The overall User Link architecture (including support for common links such as Patient Link) is illustrated in Figure 19: User Link Component Architecture. (A description for how to interpret the notation used in this diagram appears in the appendix Diagramming Conventions.)
Context Management Specification, Technology and Subject-Independent Component Architecture

Figure 19: User Link Component Architecture

Component Interfaces
AR = AuthenticationRepository
CA = ContextAgent
CD = ContextData
CM = ContextManager
CP = ContextParticipant
MA = MappingAgent
SB = SecureBinding
SD = SecureContextData
II = ImplementationInformation

Note – A context agent that is a mapping agent may implement CA instead of MA. However, CA enables more efficient agent implementations.
13.7 **User Link Context Change Process**

The process for performing a context change transaction to set the user context is the same as defined for secure links in general. However, the process may optionally include the use of an authentication repository.

The high-level events that transpire when a user signs-on are summarized in Figure 20: User Link Context Change Process. This description assumes that a user mapping agent is present and that a user digital certificate annotation agent is present. The user mapping agent is presumed to know the logon names for all users for all applications. (See Section 13.17 Populating the User Mapping Agent.) The user certification annotation agent is presumed to know the digital certificates for all users. The description omits most of the details pertaining to the surveying of the participant applications by the context manager. This process is identical to the process defined for Patient Link. (See Chapter 7)

![Figure 20: User Link Context Change Process](image)

13.8 **Designating Applications for User Authentication**

Any User Link-enabled application can serve as the means by which a user signs-on to all of the User Link-enabled applications on a desktop. To serve in this capacity, the User Link-enabled application shall provide a mechanism for establishing and authenticating the user’s identity.

The CMA does not specify an application’s user authentication mechanism, visual appearance, or implementation. The authentication mechanisms can vary among applications. Applications can be created whose sole purpose is to enable user authentication for desktops comprised of User Linked applications.
However, even though any User Link-enabled application has the potential to be used for signing on to a desktop of User Linked applications, the provider institution designates the specific application or applications it trusts for this task. Only the designated applications shall be allowed by a context manager to complete a context change transaction that involves a change to the user subject.

The one exception to this rule is that any application shall be allowed to set the user subject to empty. This is so that any application can be used to log-off from a desktop of User Linked applications. (See Section 13.13, Logging-Off and Application Termination.)

A context manager implementation-specific configuration process is used for indicating the designated applications for a particular desktop. One, several, or all of the User Link-enabled applications on a desktop can be designated for this purpose. The designated applications for a desktop can differ among desktops. It is recommend that a healthcare institution analyze the use cases for their clinical applications to determine how to best deploy User Link.

The decision criteria for a provider institution’s choice of whether to designate an application for authenticating users is based upon whether they trust the application’s security capabilities as it pertains to user authentication. For example, it might not be a good choice to designate an application that maintains user passwords in plain text (which can easily be read by unauthorized users).

### 13.9 Signing on to Applications Not Designated for Authenticating Users

A User Link-enabled application that has not been designated for authenticating users on a particular desktop shall not allow the user to sign on to the application or the desktop. The user must sign on to a designated application in order to sign on to a linked but non-designated application. The user must break a non-designated application’s link with the common context in order to sign on to just the application.

If the application has not been designated for authenticating users and it is the first to be launched on the desktop, the user must either launch an application that has been designated for authenticating users, or the user must break the link of the non-designated application. The user can then sign on to just the non-designated application.

The CMA does specify a means by which an application can determine whether it has been designated for authenticating users. See Section 15.3.9, SecureBinding (SB). This enables an application to determine whether it has been designated before a user attempts to sign on to the application. An application can use this information to present or hide its user interface user sign on controls accordingly.
13.10 **Application Behavior When Launched**

When a User Link-enabled application is launched on a desktop, it shall join the common context system established for the desktop. The application shall set its user context to match the current user context. If the application is Patient Link-enabled, it shall also set its patient context to match the current patient context.

13.11 **Access Control Lists**

Access control lists (ACL), which determine the privileges and capabilities a particular user has, are presumed to be maintained by each application. While it is desirable that there be only one centrally administered ACL, achieving this is beyond the scope of the CMA. However, before central or distributed ACL’s can be properly used it is essential that the user be authenticated. This is precisely the capability that User Link supports.

13.12 **Changing Users**

With User Link, it is advantageous for applications to support a change-user capability. This capability enables a new user to sign on without explicitly requiring that the current user first log off. There are two ways in which this can be implemented by an application:

- The application performs a single user context change transaction to establish the new user as the current user.

- The application performs a two-step process. In the first step, the current user is logged off and the user context is set to empty (to indicate that there is no user). In the second step, the new user is signed on, and the user context is set to indicate who the new user is.

The first approach is recommended because it is the simplest and the most efficient from the perspective of the context system (e.g., only one context change transaction per user change). The second approach is acceptable, however the two step process should be invisible to users.

The gestures needed to change the user, and the appearance of the application as it pertains to this capability, are not specified by the CMA.
13.13 Logging-Off and Application Termination

User Link provides applications with an easy way to enable users to:

- Terminate a specific User Linked application on the clinical desktop.
- Log off from a specific User Linked application on the clinical desktop.
- Log off from all of the User Linked applications on the clinical desktop.

There are many possible ways in which these capabilities can be realized in a common context system. The approach described in Table 1: User Link-Enabled Application Behavior for Termination and Log-Off is defined because it is simple for users to understand, yet enables design flexibility for application developers.

<table>
<thead>
<tr>
<th>User Action</th>
<th>Effect on Application That User’s Action Is Directed At</th>
<th>Effect on the Common Context</th>
<th>Effect on Other User Linked Applications on the Desktop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminate a specific User Linked application.</td>
<td>Application leaves the common context, ceases execution, and exits</td>
<td>None.</td>
<td>None.</td>
</tr>
</tbody>
</table>
| Log-off from a specific User Linked application. | Application:  
- continues to run,  
- logs the user off,  
- visually indicates that it has no user,  
- leaves common context (i.e., breaks link) | None. | None. |
| Log-off from all of the User Linked applications that are resident on the same desktop. | Application:  
- continues to run,  
- instigates a context change transaction to set the user context to empty,  
- visually indicates that it has no user,  
- continues to be a context participant. | User subject set to empty. | When the context change is completed, each application:  
- continues to run,  
- logs the user off,  
- visually indicates that it has no user,  
- continues to be a context participant. |

Table 1: User Link-Enabled Application Behavior for Termination and Log-Off

5 Terminating all of the applications on a desktop is not supported because there is no way to indicate this event via a change to the user context subject.
The basic idea is that each User Link-enabled application optionally supports gestures that enable the user to terminate the application, log off from just the application, or log off from all of the User Linked applications that are resident on the same desktop.

All User Link-enabled applications must behave properly as participants in a context change transaction, as described in Table 1. All User Link-enabled applications must be able to properly deal with the context when the user context is empty.

However, the CMA does not specify the user gestures that are needed to initiate the actions described in Table 1. The gestures may be different among applications. Further, an application may choose which action gestures, if any, it will support. For example, a particular application might not enable the user to terminate it, log off from it, or log off from the User Linked desktop.

An application that enables the user to log off shall clearly indicate that in doing so, the user will cause the application to break its link with the common context system.

There are several subtleties involved with the behaviors described in Table 1:

- Any application can set the user context to empty, including applications that have not been designated for authenticating users. This enables any application to be used for logging off from all of the User Linked applications on a desktop.

- A user might terminate the application(s) designated for authenticating users. The next user will need to relaunch one of the designated applications before being able to sign on to the User Linked desktop.

- It is conceivable that the collective capabilities of a particular set of User Link-enabled applications on a desktop result in a system that does not provide any way for the user to log off from the desktop. A site must be mindful in its choice of applications in order to prevent this from happening.

One issue with desktop log off is the treatment of “busy” applications. Busy applications affect single sign on as well as desktop log-off, and is dealt with in Section 11.6, Busy Applications.
13.14 Automatic Log-Off

An automatic log-off logs the current user off of the User Linked applications on a desktop when the user has not interacted with the applications for an appreciable period of time.

Any application can initiate an automatic log-off by performing a context change transaction that sets the user context to empty. This will have the effect of causing all of the other User Linked applications on the desktop to also log the user off. Once an automatic log-off has completed, the next user signs-on via one of the designated applications.

In contrast to a user-initiated log-off, an automatic log-off is initiated automatically by an application. The CMA does not specify an automatic log-off policy or implementation. It is an application decision as to how and when to initiate an automatic log-off.

For example, an application might monitor user interactions with the mouse and keyboard to determine whether or not the user is actually engaged in using any of the applications on the
desktop. The capability to do this depends upon the application’s implementation and the underlying desktop technology.

An application that initiates a context change transaction to affect an automatic log-off must be prepared to handle the condition in which surveyed applications are busy, or have responded with a conditional accept of the transaction. In this case the instigating application shall cancel the context change transaction. It shall not present a dialog to the user, as this could be disruptive or confusing to the user. The application may elect to initiate an automatic log-off again in the future.

It is necessary that the administrator is able to configure the behavior of automatic log-off as it pertains to a clinical desktop. Otherwise, the administrator has no control over an application whose policy for initiating an automatic log-off interferes with the users’ work.

Therefore, any application that initiates an automatic log-off shall provide a means for controlling this capability. Specifically, it shall be possible to configure that application in terms of whether the log-off it initiates is desktop-wide (and therefore affects all of the context participants), or is limited to just the application. If the automatic log-off is limited to just the application, then the application shall not perform a context change transaction when the automatic log-off interval transpires. Instead, it shall just log the user off from itself.

13.15 Reauthentication Time-out

A reauthentication time-out requires the currently signed-on user to reauthenticate herself before being allowed to continue using the applications on a clinical desktop. The time-out occurs when the user has not interacted with the desktop for an appreciable period of time. Applications maintain their internal state as the user left it prior to the time-out, but interaction with the applications cannot resume until the user has been reauthenticated.

The time-out often manifests as a screen that overlays the entire display and that provides a mechanism with which the user can reauthenticate herself. However, the CMA does not specify a reauthentication time-out policy, visual appearance, or implementation.

Any application can initiate a reauthentication time-out. However, a User Link-enabled application that does so shall be:

- responsible for enabling the user to re-authenticate herself
- configurable such that a systems administrator can enable or disable the time-out capability.

These requirements enable sites to practice the following CMA recommendation: only a User Link-enabled application that has been designated for authenticating users should be allowed
to initiate a reauthentication time-out. This enables the user to reauthenticate herself using an application that is also normally used for signing on to the clinical desktop.

This recommendation avoids the problem of forcing the user to be reauthenticated by an application not normally used for signing on, and therefore having to remember their logon name and password for the application.

Once the current user is reauthenticated, then the User Link-enabled applications resume as they were. If a different user signs on, then the User Link-enabled applications handle this as they do whenever there is a change of user.

13.16 Co-Existence with Applications Not User Link-Enabled

User Link-enabled applications will co-exist with applications that are not User Link-enabled. Users will still need to manually sign on to and log-off from each of the applications that are not User Link-enabled.

Co-existence can create confusion among users, as they might assume that all of the applications on a desktop are User Link-enabled. Training, plus visual cues documented in the HL7 context management technology-specific user interface specification documents are partial solutions. Ultimately, users will come to learn which applications are User Link-enabled, and which are not, and will adjust their use of these applications accordingly.

13.17 Populating the User Mapping Agent

The user mapping agent is conceptually similar to the patient mapping agent defined for a Patient Link common context system. For example, both types of mapping agents implement the same interface specification, MappingAgent. However, the behavior and management of the user mapping agent is substantially influenced by security considerations. Several of these considerations are described in this section. The role of the user mapping agent is illustrated in Figure 21: User Subject Context Data Mapped for Different Applications.
In order for the user mapping agent to be able to provide additional logon names for users, it must be populated with the necessary logon names. However, unlike the patient mapping agent, for which there exists healthcare standards that can be used to obtain the necessary patient data (e.g., HL7’s Admission/Discharge/Transfer messages), an equivalent means does not exist for user data. In the absence of applicable standards, the means by which a user mapping agent is populated depends upon the user mapping agent implementation.

### 13.18 Authentication Repository

The chain of trust has the potential to maximize the overall security of a common context system because the data used to authenticate a user is never passed between applications and therefore cannot be easily intercepted or spoofed. However, not passing around this data creates a problem when there are applications that require user authentication data to perform a user sign on. For example, many existing healthcare applications require the user’s password to establish sessions with their underlying databases.

The common context system therefore includes a user authentication data repository as an additional context management component. This repository enables applications to securely maintain application-specific user authentication data. The repository is used by applications...
that do not have a built-in means to easily sign on a user given only a logon name. The repository may be implemented as a distributed or centralized service.

For example, some applications obtain the user’s password from the user and then hand it off to an underlying database. The database does the actual authentication. The security capabilities of the database prevent these applications from retrieving user passwords. Therefore, it is not possible for these applications to sign on a user knowing only the user’s logon name. For these applications, an external means of maintaining user logon names and associated authentication data is required.

The authentication repository provides a way of doing this that is minimally invasive to the application. The repository is not used for authenticating users. Rather, it enables existing applications that need user authentication data to sign on the user to have a means for obtaining this data when participating in a User Link common context system.

The User Link user authentication data repository provides the capability to securely store the data that an application uses to authenticate its users. The application can use a user’s logon name to retrieve the user’s authentication data from the repository. The application can then use the authentication data to establish a user session with a database or other underlying application services.

In keeping with the spirit of the CMA, the interfaces to the authentication repository, but not its implementation, are defined. These interfaces enable an application to securely retrieve a user’s authentication data and to update this data when necessary (for example, if the application periodically requires that users change their passwords).

### 13.18.1 Repository Implementation Considerations

The repository can be implemented as a central or distributed service that services multiple applications. However, the repository shall always appear as a private service to each application. This means that an application should never be aware that there are other applications using the repository.

The user authentication data stored in the repository on behalf of an application shall be encrypted by the application prior to being communicated to the repository. The encryption technique that is used is determined by the application. The authentication data shall remain encrypted within the repository, as the repository never has the need to interpret or use this data.

The interface AuthenticationRepository enables an application to put tuples comprised of a logon name and a corresponding bit stream (representing the user’s authentication data) into the repository. This interface also enables an application to retrieve a user’s authentication data using the user’s logon name.
The means by which the repository maintains its data must be secure and shall guard against security attacks. However, the security mechanisms that are employed to achieve these objectives are an authentication repository implementation decision.

13.18.2 Populating the Repository

The authentication repository needs to be populated with the authentication data for each user for each application that it services. One way to do this is to create a batch process that loads the necessary data. However, in many cases the necessary data is inaccessible. For example, most database management systems do not provide a means for accessing the user passwords that they store.

A simpler alternative is to incrementally populate the repository. This can be accomplished by involving each of the applications that use the repository in the process of populating the repository, as follows:

• When the context manager informs the application that the user context has been set, the application obtains the logon name for the new user from the context manager.

• The application then accesses the repository to securely retrieve the user’s authentication data. The user’s logon name is supplied as the search parameter.

• If the repository cannot find the user logon name, which will be the case if the repository has not yet been populated with data for the user, then it informs the application that the logon is not known.

• The application then prompts the user to enter his/her authentication data by whatever means the application normally uses (e.g., a password dialog box).

• The application attempts to sign-on the user using whatever underlying mechanism (e.g., database) it normally uses to do this.

• If the user is successfully signed on, then the application updates the authentication repository with the user’s authentication data, using the user’s logon as the update key. The application shall encrypt the user’s authentication data prior to putting the data in the repository.

This scheme is relatively easy to implement for almost any application. It is essential, though, that the repository and its interfaces are secure, as detailed in Chapter 15.
14 Chain of Trust

This chapter defines the behaviors, algorithms, policies, and protocols that User Link-enabled applications and components must adhere to in order to properly realize the chain of trust.

14.1 User Context Change Transactions and the Chain of Trust

The major difference between a context change transaction that involves secure subjects and a transaction that involves only common subjects is support in the former for the chain of trust. Additional application and component behaviors are defined to prevent the chain of trust from being violated.

Two types of defenses are required:

- The applications and components that participate in the chain of trust must be able to authenticate each other’s identity. The objective is to prevent rogue applications or components from impersonating a real application or component as a means to manipulate the user context. Such manipulations could result in an unauthorized user gaining access to the Secure Link-enabled applications.

- The applications and components that participate in the chain of trust must be able to validate the integrity of user context data that they communicate to each other. The objective is to prevent a rogue program from modifying the data as it is passed between applications and components as a means to manipulate the user context. Such manipulation could result in an unauthorized user gaining access to the User Link-enabled applications.

Techniques for creating the chain of trust using passcodes, message authentication codes, and digital signatures are described next.

14.2 Creating the Chain of Trust

There are three general sources of mechanisms for creating the chain of trust:

- Mechanisms incorporated into existing commercially available object infrastructures, such as those based upon CORBA or COM.

- Mechanisms based upon existing commercially available secure communications infrastructures, such as the Secure Socket Layer service (SSL) or the Secure Hyper-Text Transfer Protocol (S-HTTP).
Mechanisms based upon existing widely available security building blocks, such as public key / private key encryption.

These alternatives are discussed next.

### 14.2.1 Object Infrastructures

It is conceivable that the chain of trust could be realized using the security mechanisms built into commercially available object infrastructures such as those based upon CORBA or COM. Unfortunately, these infrastructures currently employ security models that are fundamentally different from what is needed for User Link:

- Security for these infrastructures is based upon keeping track of who the user is and their respective access privileges.
- To do this requires that the user has signed on to the underlying operating system.
- However, signing on at the operating system level takes too much time. This is the very problem that User Link is trying to solve.

For example, security in Microsoft’s COM-based infrastructure is based upon tracking who the user is and what their permissions are. This means that when security is enabled for a COM interface, a COM server accepts or rejects a COM client’s access attempts based upon the privileges of the user on whose behalf the COM client is working. This does not work for User Link because a COM server (specifically, the context manager) needs to accept or reject accesses based upon which application is the COM client. The user is not relevant in this case.

It may be possible to establish a stylized approach for adapting object infrastructure security mechanisms to realize the chain of trust. However, this could make it particularly difficult to define a technology-neutral specification for the chain of trust. This could result in different User Link architectures for different technologies. This is counter to the overall CMA objective of technology-neutrality.

### 14.2.2 Secure Communications Protocols

User Link-enabled applications and the various CMA components could communicate using a secure communications protocol, such as the Secure Sockets Layer (SSL) service. SSL enables secure (i.e., encrypted) transmission of data between a client and a server. It also enables a client to authenticate a server (and a server to authenticate a client).

SSL uses the RSA public key encryption system for authentication and for data integrity and confidentiality. Of interest for the chain of trust is the SSL capability for clients and servers to authenticate each other. An SSL server uses its private key to create a digital signature. Public keys are issued to prospective clients. The public key is used by the client to
authenticate the server by decoding the server’s signature. Only a signature that has been encoded using the server’s private key can be (easily) decoded via the server’s public key.

For example, in the chain of trust, an SSL connection would be established between an application that has been designated for authenticating users and the context manager. In this scenario, the application is an SSL server, while the context manager is an SSL client.

SSL and its secure communications counterparts, such as S-HTTP, provide off-the-shelf mechanisms for implementing the chain of trust. However, this technology has not been integrated with popular object infrastructures, such as those based upon COM or CORBA.

While secure communication services could provide a means for implementing the chain of trust, the practical implications of using multiple communications technologies within the User Link architecture are a cause for concern. For example, it could become overly complicated to have some communications be via COM or CORBA interfaces, while other communications use SSL or S-HTTP.

Further, the chain of trust generally does not require confidentiality. For example, the User Link architecture does not require that sensitive data, such as a user’s password, be communicated between applications. Secure communication channels are overkill and are not a good fit for User Link.

### 14.2.3 Security Building Blocks

The security building blocks that are available on most popular operating systems can form the basis for realizing the chain of trust. The two building blocks of particular interest are:

- Digital signatures.
- Secure (or one-way) hashing.

Digital signatures, which cannot be easily forged, are typically used by people as a means to authenticate each other’s identity whenever they communicate electronically. However, a digital signature also enables an application or component to identify itself in a way that can be authenticated whenever it communicates with another application or component.

Digital signatures are formed using public key / private key encryption techniques. While these techniques enable encryption, they also enable the formulation of digital signatures. An application or component formulates its digital signature using its private key and sends the signature along with the data that it wants to share. The recipient of a signed message applies the sender’s public key to the signature to authenticate the sender and to verify the integrity of the data that was sent.
There are several public key/private key algorithms and related standards. Commercial implementations of many of these algorithms are available in a variety of technologies. RSA is an example of an algorithm that has been widely implemented.

A secure hash function is used for producing a unique numeric surrogate from an arbitrary data stream. It is improbable that two different data streams will yield the same hash value. A secure hash function is an essential part of the infrastructure needed to support the use of digital signatures.

Specifically, a secure hash function enables the efficient computation of a digital signature. A secure hash function also plays a role in enabling public keys to be reliably distributed. It is essential that the holder of a public key is able to determine who (or what) the key belongs to. Otherwise an impostor could present its own public key while claiming to be someone or something that it is not. The holder of the public key would mistake subsequent communications as coming from a valid source when in fact it came from an impostor.

There are several secure hashing algorithms and related standards. Commercial implementations of many of these algorithms are available in a variety of technologies. MD5 is an example of an algorithm that has been widely implemented.

Taken together, digital signatures and secure hashing could be used in the chain of trust as the means for User Link-enabled applications and User Link components to authenticate each others’ identity each time they communicate. This capability is fundamental to the establishment and maintenance of the chain of trust.

To accomplish this, a digital signature would be explicitly included as a method parameter for each CMA-specified interface that requires this level of security. The use of digital signatures enables the specification of a system that has the desired User Link semantics and that can be readily implemented using existing security standards and technology.

Creating a system that employs digital signatures for applications and components is simpler than creating a signature-based system for users. This is because the population of applications and User Link components that requires signatures is small compared to the number of users of the system. Further, the population of applications and User Link components does not change nearly as often as the user population. The result is that the work required to create and maintain the chain of trust is substantially less than would be the case if user signatures were required.

Another advantage of digital signatures is that they can be used to ensure the integrity of any data communicated during interactions among and between User Link components and User Link-enabled applications. The recipient of the data can use the signature to determine if the data has been tampered with between the time it was sent and the time it was received.
Method-based digital signatures fit well with the component-based Context Management Architecture. For example, realizing the chain of trust in this manner enables a technology-neutral specification for the chain of trust. This is because the approach can exploit capabilities common to public key / private key implementations that are commercially available in multiple technologies. Further, the ways in which digital signatures are used can be arranged to achieve the desired security behaviors needed for User Link.

The trade-off is that more effort is required to architect the chain of trust than would be the case if a standard “off-the-shelf” component-based solution was available. This trade-off is viewed as acceptable. Therefore the approach pursued in the CMA is to use method-based digital signatures as the basis for the chain of trust.

14.2.4 Security Attacks On the Chain Of Trust

The primary challenge for realizing the chain of trust is minimizing the likelihood that an intruder is able to violate the chain of trust to obtain access to a User Link-enabled application. This violation could occur if a rogue program was able to set the user context to represent a user who either has not been authenticated, or who is different from the user who has been authenticated.

The chain of trust based upon the security building blocks described in Section 14.2.3 \[Security Building Blocks\] defends against the security attacks described in the table below, all of which are directed at manipulating the user context. Refer to \[Figure 20: User Link Context Change Process\] for the specific trust relationships:
Table 2: Chain of Trust Attacks and Defenses

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attempt to impersonate an application in order to set the user context (Step #2).</td>
<td>An application presents its signature to the context manager in order to set the user context. The context manager uses the signature to authenticate the application to ensure that has been designated for authenticating users.</td>
</tr>
<tr>
<td>Attempt to impersonate the context manager so that the user context that the user mapping agents sees, and therefore maps, is bogus (Step #3).</td>
<td>The context manager presents its signature to the mapping agent when the mapping agent gets the user context data from the context manager. The mapping agent uses the signature to authenticate the context manager.</td>
</tr>
<tr>
<td>Attempt to impersonate the user mapping agent as a means to set bogus user logon names within the user context (Step #3).</td>
<td>The mapping agent presents its signature to the context manager when it sets user context data. The context manager uses the signature to authenticate the mapping agent.</td>
</tr>
<tr>
<td>Attempt to impersonate the context manager so that the user context that a participant application sees is bogus (Step #5).</td>
<td>The context manager presents its signature to the participant application when the application gets the user context data from the context manager. The application uses the signature to authenticate the context manager.</td>
</tr>
<tr>
<td>Attempts to impersonate the authentication repository as a means to obtain user authentication data from an application (Step #6b).</td>
<td>The application encrypts the user authentication data using the authentication repository’s public key before providing the data to the repository. Only the real authentication repository can decrypt this data. Further, the application pre-encrypts the data using an application-specific encryption scheme. The data remains encrypted even when stored inside the repository.</td>
</tr>
<tr>
<td>Attempt to impersonate an application as a means to obtain user authentication data from the authentication repository (Step #6b).</td>
<td>An application must present its signature to the authentication repository when it gets user authentication data from the repository. The repository uses the signature to authenticate the application. Further, the application encrypts the authentication data before storing it in the repository. Only the application that encrypted the data can subsequently decrypt it.</td>
</tr>
</tbody>
</table>

The chain of trust does not necessarily need to defend against every type of attack, including attacks to gain access to the user’s logon name (i.e., Step #4). A user’s logon name is easy to guess or obtain, and in the absence of user authentication data (e.g., a password) a logon name does not provide a means for gaining access to a system.

The chain of trust also does not defend against applications that do a poor job of authenticating users (i.e., Step #1). Provider institutions must ensure that the applications they designate for authenticating users meet their security needs.
Other types of attacks that are not defended by the chain of trust can result in a denial of service, which may cause a common context system to function improperly. For example, a rogue program might continually invoke context manager methods, causing the context manager’s performance to degrade while it services these invocations.

These programs do not breach security in terms of enabling unauthorized access to User Link-enabled applications, but they do result in inconveniences for users of the system. In general it is extremely hard, and can be quite costly, to defend against denial of service attacks.

The most effective preventatives for denial of service attacks begin with physical security, in which a malicious user is denied access to any of the computers within a system. Without access to the system, a malicious user will have a much harder time installing rogue programs. Physical security is strongly encouraged, but it is beyond the scope of the CMA to specify the necessary measures.

Additional potential limitations of the chain of trust are described in Section \[4.2.5\] Chain of Trust Implementation Limitations

### 14.2.5 Chain of Trust Implementation Limitations

A secure implementation of the chain of trust requires that the User Link components (i.e., context manager, applications, mapping agent, authentication repository) all have a robust way of authenticating each other’s identity. Providing this capability requires the use of underlying operating systems primitives, including file access privileges and memory protection mechanisms.

Not all operating systems implement these security primitives to the same degree of robustness. The approach for implementing the chain of trust described below is therefore fundamentally limited by the capabilities (or lack thereof) of the underlying operating system upon which a User Link system is deployed.

In particular, Windows NT and most Unix-based operating systems provide the necessary primitives. User Link systems deployed on these operating systems will offer robust security capabilities. In contrast, Windows 95 and Windows 98 lacks many of the necessary primitives. User Link systems deployed on this operating system will offer useful capabilities, but the systems will not be any more secure than native Windows 95/98.

### 14.3 Digital Signatures and CMA Components

Digital signatures created using a public key / private key encryption system are incorporated into the component interfaces defined for User Link-enabled applications and components. In the chain of trust these signatures (and corresponding keys) are not associated with a user,
but rather with an application or component. The signatures and keys for a particular
application are the same independent of who the user is.

Several of the methods defined for the existing context manager interfaces already require
that applications identify themselves (e.g., ContextData::SetItemValues ). The participant
coupon, which is an integer, is assigned by the context manager to an application when it
joins a common context system (via ContextManager::JoinCommonContext). This coupon is
subsequently used by the application to identify itself when it calls a context manager
method that requires application identification.

The methods requiring applications to identify themselves do so to enforce the correct
behavior of a common context system. For example, only the application that instigated a
context change transaction or a mapping agent can set context data. Similarly, only the
instigating application can end the transaction in progress.

However, the use of a participant coupon is not intended to be a security mechanism. For
example, a rogue application can impersonate a valid application by obtaining (or even
guessing) the value of the valid application’s coupon. Coupons are simply to enable the
context manager to identify the applications it is dealing with.

An elaboration of the coupon approach is to use digital signatures as a means for applications
to identify themselves in a manner that can be authenticated. It is relatively straightforward
to use digital signatures in addition to coupons whenever it is necessary to authenticate an
application or component.

Based on this approach, CMA interfaces are defined that enable the establishment of the
necessary signature-based security relationships among and between applications and context
management components. Additional CMA-defined interfaces subsequently enforce these
security relationships as applications and components interact during the course of a context
change transaction.

### 14.3.1 Public Key / Private Key Encryption as a Means for Generating Signatures

Providing applications with digital signatures requires that each application or component
that is to be trusted is assigned a public key and private key based upon an algorithm such as
RSA. The private key is used to create a digital signature. The corresponding public key is
used to verify the signature.

For example, an application supplies its participant coupon and its signature to the context
manager whenever it performs a context manager method that requires the context manager
to authenticate the identity of the application and validate the integrity of the data sent by the
application.
A digital signature is formed by applying a secure hash function (alternatively known as a one-way hash function) to the data that is to be transmitted. The resulting hash value is referred to as the message digest, as it is a numeric surrogate for the plain-text message. It is computationally improbable that two messages will produce the same hash value.6

The message digest is then encrypted by the sender using its private key. The digest can only be decrypted using the sender’s public key. In other words, any party holding the sender’s public key can authenticate that the message came from the sender and that the data sent was received in tact.8

The encrypted hash value enables the sender of the data to ensure that the receiver of the data can authenticate the sender’s identity. The receiver uses the same secure hash function as the sender to perform its own computation of a hash value using the data it received. Note that the data was not encrypted. Just the hash value computed from the data was encrypted.

The receiver compares the hash value it computed with the value it decrypted. The encrypted hash value can only be successfully decrypted using the public key that matches the sender’s private key. If the hash values match, then the data sender’s identity has been confirmed, and the integrity of the data has been validated.

If the hash values do not match, then either the data was tampered with between the time it was sent and was received, or the sender is not who it claims to be.

The algorithm for creating the hash value must be compatible with the public key / private key scheme that is employed. For example, if RSA is the public key / private key scheme that is used, then an RSA-supported hashing algorithm (e.g., MD5, SHA-1) must be employed to create the hash value. When the signature is computed in this manner, authenticity and data integrity can be verified.

The specific secure hash algorithm and the public key / private key scheme that is employed is technology-specific. Each of the HL7 Context Management Technology Mapping Specifications indicates the secure hash algorithm public key / private key scheme that is needed for a particular technology-specific implementation.

---

6 When a secure hash function is used, it is also computationally infeasible to invert the computed hash value. Specifically, given the secure hash function $f$ and input value $x$, $f(x)$ is relatively easy to compute. However, even knowing $f$ it is infeasible to compute $x$ given $f(x)$.

7 The signing of a message digest rather than of the plain-text message is a performance expediency. A digest is typically several bytes in size, whereas the message represented by a digest can be of arbitrary size. It is generally faster to encrypt the digest rather than the entire message.

8 This is the inverse of the process used to send a secret message, in which the sender encrypts data with the intended recipient’s public key. Only the holder of the private key can decrypt the data.
The overall process for signing a message is illustrated in Figure 22: Signing A Message.

![Diagram of Signing A Message]

**Figure 22: Signing A Message**

### 14.3.2 Incorporation of Signatures into the Context Management Architecture

Digital signatures are incorporated in the Context Management Architecture to enable authentication between User Link-enabled applications and User Link components. For example, digital signatures enable the context manager to authenticate the identity of any application that performs a context manager method. The context manager can also ensure the integrity of the parameter values that it received from the application.

The context manager accomplishes this by computing a hash value from the input parameters it receives from the application. To obtain the application-computed hash value from the signature the context manager must use the same public key / private key scheme as the application. The context manager must also use the same hash algorithm as the application.

The context manager compares the hash value it computes to the hash value it has obtained by decrypting the application’s digital signature. If the two hash values match, then the method invocation is authentic and data integrity is ensured.

Otherwise, there has been a breach of security: either the method was invoked by an impostor of the application, and/or the parameter values provided by the application were tampered with after they were sent but before they were received by the context manager. The context manager rejects the method invocation.
To be more specific, for the context manager method SecureContextData::SetItemValues, the hash value would be computed using the value of the participant application’s coupon (i.e., input parameter `participantCoupon`), current context change transaction coupon (i.e., input parameter `contextCoupon`), the names of the items whose values are to be set (i.e., input parameter `itemNames`), and the values for these items (i.e., input parameter `itemValues`).

The use of a hash in forming a signature is illustrated Figure 23: Forming Signature Using Method Parameters.

XXX’s signature is the result of XXX using its private key to encrypt a hash value computed using the parameter values it provides in the call to `SetItemValues` ... all of the applications and the context manager use the same public key/private key scheme for generating signatures. They must also use the same hash algorithm.

XXX’s signature is the result of XXX using its private key to encrypt a hash value computed using the parameter values it provides in the call to `SetItemValues` ... all of the applications and the context manager use the same public key/private key scheme for generating signatures. They must also use the same hash algorithm.

14.3.3 Computing a Digital Signature

Secure hash algorithms use a character string as the representation of the data value upon which a hash value is to be computed. Therefore, parameter values that are to be protected

---

9 This coupon denotes the current context change transaction, not the application. Each context change coupon is unique over the execution lifetime of a particular context manager.
from tampering during a method invocation must be converted to character strings. These strings must then be concatenated to form a single string. It is the concatenated string that is used to compute the hash value.

The rules for concatenation are as follows. These rules take into account the fact that the mapping of CMA interfaces to specific technologies may alter the order in which method parameters are declared and/or may require additional technology-specific parameters. The rules ensure that the process for creating signatures is invariant across technologies:

- The architectural specification for each method that is to be signed will define which method parameters must be protected from tampering, and are therefore to be used in formulating the signature.
- The architectural specification for each method that is to be signed will define the order in which the string representations of the parameters are to be concatenated.
- The string representation of an array parameter starts with the first element in the array and ends with the last element in the array.
- A parameter or array element whose value is null or empty is omitted from the string.
- An array that does not contain any elements (i.e., the array length is zero) is omitted from the string.
- Delimiters are not required because there is no need to parse the string.

For example, the concatenated string that might be produced based upon the example in Figure 23: Forming Signature Using Method Parameters would look like:

```
172User.id.logon.3M_Clinical_Workstationrobs9789
```

In another example, where the value of the context item “logon” is null, the concatenated string would look like:

```
172User.id.logon.3M_Clinical_Workstation9789
```

In a final example, where the context items are:

- User.id.logon.3M_Clinical_Workstation = “robs”
- User.co.GivenName = “Robert Seliger”

The concatenated string would look like:
The rules for representing various data types as character strings are specified in Section 15.2.9, Representing Basic Data Types as Strings.

Finally, once the hash value has been computed, encrypting the hash value with the sender’s private key generates the digital signature.

14.3.4 Public Key Distribution

Public key distribution is the process by which an entity, such as the context manager, makes its public key available to the other entities, such as an application, that need to use the key. This process must ensure that a receiving entity can reliably establish the identity of the entity that created the key. If this is not accomplished then it is possible for a rogue entity to impersonate a valid entity by representing the valid entity’s public key as its own.

In contrast, private keys are not distributed, but remain the secret of the owner of the corresponding public key. A discussion about protecting private keys appears in Section 14.3.4.3, Protecting Private Keys.

There are a variety of ways that keys can be distributed, including via a certificate authority. However, the approach chosen for the CMA minimizes the amount of infrastructure that is required to create a User Link solution, yet is upwards compatible with more elaborate approaches.

Specifically, public keys are exchanged as part of a dynamic process that occurs each time a User Link-enabled application or User Link component is launched. This approach enables a high-degree of security while minimizing the effort and cost to develop and deploy User Link solutions.

A two-step binding process is used to dynamically distribute an application’s public key. The process depends upon the use of secret passcodes that are assigned to user Link-enabled applications (specifically, applications that are capable of being designated for authenticating users) and User Link components. An application or component uses its passcode to prove its identity when it presents its public key. A passcode is a complex, arbitrary alphanumeric string.

A passcode is not actually transmitted when a secure binding is established. Instead, a secure hash function is used to produce a message authentication code. A message authentication code

10 Not all applications need a public key. Applications that need public keys are those that are designated for authenticating users, and those that use the authentication repository.
code is a secure hash value produced from a data stream that consists of data that is openly communicated between two parties, and “secret” data that they both know but do not openly communicate. In the CMA, a passcode serves as the shared secret.

The binding process involves a “bindee” and a “binder.” In order to bind, a bindee must have a passcode. Both the bindee and the binder must have knowledge of the passcode. The means for providing the bindee and binder with a passcode are not specified in the CMA. However, requirements and guidelines are described in Section [4.3.4.1, Passcode Generation].

The following table describes the relationships between User Link-enabled applications and User Link components in terms of the secure binding process:

<table>
<thead>
<tr>
<th>Bindee</th>
<th>Binder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context Participant Application</td>
<td>Context Manager</td>
</tr>
<tr>
<td>Context Participant Application</td>
<td>Authentication Repository</td>
</tr>
<tr>
<td>Mapping Agent</td>
<td>Context Manager</td>
</tr>
</tbody>
</table>

The bindee initiates the binding process with the binder. The bindee assumes it knows the identity of the binder, but will prove the binder’s identity as part of the binding process. Similarly, the binder will establish the identity of the bindee as part of the binding process.

The following interactions then occur:

1. The bindee symbolically identifies itself to the binder. The binder uses this information to locate the binder’s copy of the bindee’s passcode. The passcode is not transmitted by the bindee.

2. The binder sends back its public key, and a message authentication code. This code is a secure hash value computed from a data stream formulated from the binder’s public key and the binder’s copy of the bindee’s passcode.

3. The bindee uses the public key it has received and its copy of its passcode to formulate a data stream from which it also computes a secure hash value. (The hash algorithm it uses must be the same as the one that the binder used.) The bindee compares the resulting hash value to the message authentication code. If the two match, then the binder is who it claims to be and the public key received by the bindee indeed belongs to the binder.
4. The bindee again identifies itself to the binder and sends its public key, along with a new message authentication code. This code is a secure hash value computed from a data stream formulated from the bindee’s public key and the bindee’s copy of its passcode.

5. The binder uses the public key it has received and its copy of the bindee’s passcode to formulate a data stream from which it also computes a secure hash value. (The hash algorithm it uses must be the same as the one that the bindee used.) The binder compares the resulting hash value to the message authentication code. If the two match, then the bindee is who it claims to be and the public key received by the binder indeed belongs to the bindee.

An application requires a passcode for binding with the context manager. This passcode is a secret known only to the application and the context manager.

An application also requires a passcode for binding with the authentication repository. This passcode is a secret known only to the application and the authentication repository. An application that binds to both the context manager and the authentication repository shall use different passcodes for each binding.

14.3.4.1 Passcode Generation Requirements

Passcodes are similar to passwords used by people. However, because passcodes are only used by computer programs, they can be much longer and complex than passwords typically are. This makes passcodes extremely hard to guess, even when brute force techniques are employed.

An application passcode shall be a randomly generated character string comprised of no less than one hundred (128) characters and no greater than two-hundred fifty-six (256) characters. A passcode shall only be comprised of alphanumeric characters, as well as the underscore (_) and dash (-) characters. A passcode shall not contain white space (e.g., tabs, spaces). A passcode shall be arbitrary but shall not contain any words or phrases.

The BNF for a passcode appears below:

```plaintext
SPACE = " "
UNDERSCORE = "_"
PERIOD = "." 
DASH = "-"
ALPHA = "a-zA-Z"
NUMERIC = "0-9"
POUND = "#"

passcode = [ ALPHA | NUMERIC | UNDERSCORE | DASH ] *
```
An application’s passcode may be generated such that the same passcode is used for every instance of the application everywhere. This is the least secure means of generating passcodes, because a security breach affects every instance of the application.

An application’s passcode may be generated such that the same passcode is used for every instance of the application at a particular site. This is a moderately secure means of generating passcodes, because a security breach is at least limited to a particular site.

An application’s passcode may be generated such that a unique passcode is used for each desktop upon which the application is used. This is the most secure means of generating passcodes because a security breach is limited to a single desktop. This is the recommended approach.

14.3.4.2 Protecting Passcodes

Passcodes must remain secret. There are numerous ways in which this can be achieved. The specific approach is left as an implementation decision for applications and the various context management components.

However, the following approach is recommended for applications. The assumption is that any application that is used to authenticate users probably uses a server to maintain user account and authorization information. The application might be organized using a client/server architecture, or a web server architecture.

The principle challenge is how to create an application such that the portion of the application that serves as a context participant has a secure means to store and retrieve its passcode. In the case of client/server systems, an approach could be to store the passcode on each clinical desktop upon which the client has been loaded. In web systems, an approach could be to transmit the passcode from the web server to the desktop. Both of these approaches introduce substantial security risks that would require great effort to defend against.

An alternative is for an application to store its passcode in a server, where it can be more readily protected (including literally placed under lock and key). This could be the application’s database server, or it could be a separate server whose specific role is to securely maintain passcodes.

The server would never actually transmit the passcode. Rather, it would be responsible for verifying message authentication codes received by the application. It would also be responsible for computing the application’s message authentication code.

In this approach, the server must be able to authenticate the identity of the application. The server must also be sure that the data it sends and receives from the application is not tampered with while it is in transit. This implies that the application must have the means for
establishing a trusted relationship with the server in a manner somewhat akin to the relationship the application establishes with the context manager or authentication repository.

There are many ways in which the necessary relationship can be implemented. However, because this relationship does not involve interoperation between applications, and because the optimal approach depends heavily upon the architecture and design of the application, a single approach is not specified. Instead, the approach for the server-based maintenance of an application’s passcode is left as an application design exercise.

14.3.4.3 Protecting Private Keys

The key distribution process described in Section 14.3.4, Public Key Distribution, does not prescribe when keys are created. However, once created, a private key must remain the secret of its owner for as long as it is in use.

It is possible to statically create a public key / private key pair for an application or component. However, this approach requires the use of a persistent store within which the public key / private key pair are housed when the application or component is not executing. If such a store were used, it would need to be defended against security attacks. This can be accomplished, but at the cost of adding complexity to applications or components.

The recommended alternative approach is for an application or component to dynamically create its key pair when launched. This enables the keys to be kept in memory, and avoids the complexity of using a persistent store. While it is conceivable that an in-memory private key could be accessed by an intruder, most contemporary operating systems enable a process to prevent other processes from reading its memory.

14.3.5 System Configuration Requirements

The system configuration capabilities necessary in order to deploy a User Link system are summarized as follows:

- The context manager shall provide a means for entering the symbolic names of the applications that have been designated for authenticating users. The names of designated applications may be established on a site-wide basis or a per-desktop basis. It shall not be possible for anyone but the site’s system administrator to modify the names known to a context manager.

- The context manager shall provide a means for entering the symbolic name and corresponding passcode for each application that has been designated for authenticating users at a particular site. This process shall be performed such that the passcode remains a secret known only to the application, the context manager, and
perhaps the system administrator who conveys the information from the application to the context manager.

- The context manager shall provide a means for entering the symbolic name and corresponding passcode for the user mapping agent used at a particular site. This process shall be performed such that the passcode remains a secret known only to the user mapping agent, the context manager, and perhaps the system administrator who conveys the information from the application to the context manager.

- The authentication repository shall provide a means for entering the symbolic name and corresponding passcode for each application that uses the authentication repository at a particular site. This process shall be performed such that the passcode remains a secret known only to the application, the authentication repository, and perhaps the system administrator who conveys the information from the application to the authentication repository.

- Applications capable of being designated for authenticating users, and the user mapping agent, shall provide a means of either obtaining a passcode or for entering a passcode. This process shall be performed such that the secret passcode remains a secret known only to the application or user mapping agent, the context manager, and perhaps the system administrator who conveys the information from the application or user mapping agent to the context manager.

There are numerous ways in which these capabilities can be implemented. It is beyond the scope of the CMA to specify these capabilities. The specific approaches are left as an implementation decision for applications and the various context management components.

14.3.6 Defending Against Replay Attacks

In a replay attack, an intruder captures valid messages that have been previously communicated and retransmits them at a later time in the hope of violating a system.

For example, an intruder might capture a message that enables a user to log on. Even though the intruder might not be able to read the message (it might be encrypted), the intruder might be able to “replay” the message at later time in order to gain access to the system. In this case, the intruder would be able to log on as the user whose actions resulted in the transmission of the original message.

The general approach for defending against replay attacks is to include a “nonce” in each message. The nonce is simply a number that is different each time a message is sent, and is used in computing the hash value for a message. The recipient of a message can keep track of nonces it has seen, and simply reject messages that contain previously seen nonces.
In the CMA, context change coupons in conjunction with the recommend approach of

dynamically-generated public key/private key pairs (see Section 14.3.4.3 Protecting Private

[Keys]) defend against replay attacks.

A context change coupon serves as a nonce whose uniqueness is ensured while a context
management system is active (i.e., from the time the first participant joins to the time the last
participant leaves). Dynamically-generated keys ensure that signed messages can only be
authenticated while a context management system is active. Signed messages from earlier
activations of the system are meaningless. Together, the use of context change coupons as
nonces and dynamically generated keys provide a strong defense against replay attacks.

14.4 Trust Relationships

This section specifies application and component behaviors for realizing the chain of trust.

14.4.1 Trust Between Applications and Context Manager

A Secure Link-enabled application shall obtain a reference to the context manager’s
principal interface from the interface reference registry. The application shall interrogate this
interface to obtain a reference to the context manager’s SecureBinding interface.

A Secure Link-enabled application shall establish a secure binding with the context manager,
per Section 14.3.4 Public Key Distribution after it has joined the common context system
but before it instigates any user context change transactions. This ensures that the
application:

- is communicating with the real context manager,
- has obtained the real context manager’s public key,
- has provided the context manager with its public key.

A Secure Link-enabled application shall always create a digital signature to sign the context
manager methods it invokes in order to set context data that includes secure subject context
items. This enables the context manager to authenticate the application to ensure that the
application is allowed to set the data, and to ensure the integrity of the communicated context
data items.

A Secure Link-enabled application may also need to create a digital signature to sign the
context manager methods it invokes in order to get context data that includes secure subject
context items. This enables the context manager to authenticate the application to ensure the
that the application is allowed to get the data.
The context manager shall create a digital signature to sign return values it communicates to an application whenever these values include secure subject context items. This enables the application to authenticate the context manager, and to ensure the integrity of the communicated context data items.

All other interactions between applications and the context manager do not need to follow these rules.

14.4.2 Trust Between Context Manager and a Context Agent

A context agent shall obtain a reference to the context manager’s principal interface from the interface reference registry. The context agent shall interrogate this interface to obtain a reference to the context manager’s SecureBinding interface.

The context agent shall establish a secure binding with the context manager, per Section 14.3.4 Public Key Distribution before it accesses any user context data. This ensures that the context agent:

- is communicating with the real context manager,
- has obtained the real context manager’s public key,
- has provided the context manager with its public key.

The context agent shall create a digital signature to sign the context manager methods it invokes in order to set context data that includes user subject context items. This enables the context manager to authenticate the context agent, and to ensure the integrity of the communicated context data items.

The context agent may also need to create a digital signature to sign the context manager methods it invokes in order to get context data that includes secure subject context items. This enables the context manager to authenticate the context agent to ensure that the agent is allowed to get the data.

The context manager shall create a digital signature to sign return values it communicates to the context agent whenever these values include secure subject context items. This enables the context agent to authenticate the context manager, and to ensure the integrity of the communicated context data items.

All other interactions between the context manager and the context agent do not need to follow these rules.
14.4.3 Trust Between Applications and Authentication Repository

A User Link-enabled application shall obtain a reference to the authentication repository’s principal interface from the secure registry. The application shall interrogate this interface to obtain a reference to the authentication repository’s SecureBinding interface.

A User Link-enabled application shall establish a secure binding, with the authentication repository, per Section [4.3.4] Public Key Distribution after it has joined the common context system but before it instigates any user context change transactions. This ensures that the application:

- is communicating with the real authentication repository,
- has obtained the real authentication repository’s public key,
- has provided the authentication repository with its public key.

A User Link-enabled application shall create a digital signature to sign the authentication repository methods it invokes in order to set user authentication data. This data shall also be encrypted by a means chosen by the application, and then encrypted again upon communication using the authentication repository’s public key. The repository shall decrypt the data using its private key only when it needs to service a valid application request to retrieve the data. The repository shall never decrypt the data from its application-specific encrypted form.

This enables the authentication repository to authenticate the application, to ensure the integrity of the communicated authentication data, to keep the authentication data confidential when it is communicated, and to defend against intrusions into the repository to obtain user authentication data.

The authentication repository shall create a digital signature to sign user authentication data it communicates to an application. User authentication data that is communicated back to an application shall remain encrypted as it was when provided by the application. This data shall be encrypted again upon communication using the application’s public key.

This enables the application to authenticate the authentication repository, to keep the authentication data confidential when it is communicated, and to ensure the integrity of the communicated user authentication data.

All other interactions between applications and the authentication repository do not need to follow these rules.
14.5 Chain of Trust Interactions

The detailed interactions for several use cases involving the chain of trust are illustrated below. A description for how to interpret the notation used in these diagrams appears in Appendix I. The following additional notation is used:

- The character “|” indicates the concatenation of two strings, for example, “qrs|xyz” to form “qrsxyz”.
- XXSignature(a|b|c) indicates the digital signature for XX. The signature is formed by applying a one-way hash function to the parameter values a, b, and c, and then encrypting the resulting hash value using XX’s private key.
- XXPublicKey(abcd) indicates that the data “abcd” is encrypted using the public key for XX.
- XXEncrypt(abcd) indicates that the data “abcd” is encrypted using an encryption scheme chosen by XX.
- Hash(abcd) indicates a value produced by applying a one-way hash function to the data “abcd”.
- The abbreviation ZZ represents application ZZ, CM represents the context manager, AR represents the authentication repository, and MA represents the user mapping agent.
Repository has no user data for "robs" so Application ZZ queries user for his authentication data. Application ZZ then populates repository with encrypted user authentication data.

Interaction Diagram 17: Populating Authentication Repository with User Authentication Data
Interaction Diagram 18: User Link Context Change Transaction
15 Interface Definitions

It is assumed that an underlying technology infrastructure that supports distributed objects is used to implement a common context system, although a specific technology is not assumed. However, the capabilities of Microsoft’s COM-based Automation technology are considered as a baseline. This implies that the architecture must work well within the constraints of Microsoft Automation, including issues that pertain to performance and supported data types.

An abstract set of CMA component interface definitions is described below. These interfaces are defined using a precise and concise interface definition language (IDL) created for specifying the CMA. This IDL is not meant to be a comprehensive interface specification language. Only the capabilities that are required for specifying CMA component interfaces are included in the IDL.

A CMA-specific IDL is used because existing interface specification languages have direct or indirect ties to specific technologies. For example, OMG’s IDL implies that the interfaces are implemented using CORBA-based technology. Microsoft’s MIDL requires that the interfaces are implemented using COM/DCOM technology. The use of these specification languages confuses and possibly compromises the technology-neutrality of the CMA specification.

Experience has shown that the interface constructs represented in IDL defined below can be easily mapped to interfaces that can be implemented using a specific technology such as ActiveX, CORBA, Java, or HTTP. The mapping for each specific technology appears in a separate Context Management specification document.

15.1 Interface Definition Language

The interface definition language (IDL) used in this document enables specifying the following facts about a component interface:

- The interface’s symbolic name.
- The set of component properties and methods that can be accessed via the interface.
- The name and data type of each property, and optional restrictions (e.g., read-only).
- The names and data types for each method’s input and outputs.
- The names and data content for each method’s exceptions.

The IDL also defines a set of simple data types and the capability to represent sequences of these types.
In the following sections, IDL reserved words are shown in bold font. Identifiers are shown in italics. An identifier is an alphanumerical string that starts with an alphabetic character.

### 15.1.1 Interface Definition Body

The body of an interface definition creates a lexical scope distinct from all other interface definitions. The body of an interface is specified as:

```idl
interface interfacename { ... }
```

*Interfacename* is the symbolic name of the interface. The curly brackets delimit the scope of the interface’s body.

The body of an interface begins with the declaration of any exceptions that can be raised by methods defined for the interface. The details of declaring exceptions are discussed later.

The properties that can be accessed through the interface are listed next. A property is a data value that can be read or set via the interface:

```idl
datatype propertyname
```

*Datatype* is the data type for the property. The type is one of the simple types defined below, as denoted by the appropriate IDL reserved word.

*Propertyname* is the symbolic name of the property. A property’s name must be distinct as compared to the names of other properties, methods, and exceptions defined within the same lexical scope.

Properties can also be sequences. Sequences are described below.

Properties can be restricted to read-only:

```idl
readonly datatype propertyname
```

The value of a read-only property can be read, but not set, via the interface.

Finally, the methods are listed:

```idl
methodname inputs ( ... ) outputs ( .... ) exceptions ( ... )
```

*Methodname* is the symbolic name of the method. A method’s name must be distinct as compared to the names of other properties, methods, and exceptions defined within the same lexical scope.

The method’s inputs, outputs, and exceptions follow the method’s name. If a method does not have any inputs, outputs, or exceptions, then only white space shall appear between the appropriate set of parentheses.
Each input and output is defined as:

```
datatype name
```

*Datatype* is the data type for the input or output. The type is one of the simple types defined below, as denoted by the appropriate IDL reserved word. In an actual interface definition, the appropriate IDL reserved word is used to indicate the type. Inputs and outputs can also be sequences. Sequences are described below.

*Name* is the symbolic name of the input or output. The name of inputs for a method must be distinct for the method. The name of each output for a method must be distinct for the method.

Multiple inputs and outputs are separated by a comma.

Exceptions are listed only by their name. Multiple exceptions are separated by a comma.

### 15.1.2 Simple Data Types

The following simple data types are supported. The reserved words used to indicate each type are shown:

- **byte**: Eight uninterpreted bits
- **short**: 16-bit signed integer
- **long**: 32-bit signed integer
- **float**: 32-bit floating point number
- **double**: 64-bit floating point number
- **boolean**: Indicates true, or false
- **string**: A string of characters
- **date**: A specific year/month/day/time, with a precision of one second, and including the time zone
- **type**: An enumeration that denotes each of these data types (except type) as well as the special types null (valid value not known) and empty (data type not known)
- **variant**: A tagged union of all of these data types (including type and variant)

The concrete representations of these data types are not defined. They depend upon the interface implementation technology.
15.1.3 Exception Declaration

An exception declaration introduces an exception that can be raised by one or more of the methods defined for the interface within whose lexical scope the exception declaration appears. Each exception declaration indicates the exception name and an optional set of data values. The name denotes the exception and the data values provide additional run-time information about the reason for the exception.

An exception declaration is specified as:

```
exception name { ... }
```

*Name* is the symbolic name of the exception. An exception’s name must be distinct as compared to the names of other properties, methods, and exceptions defined within the same lexical scope.

Exception data values are specified as:

```
datatype name ;
```

*Datatype* is the data type for the exception value. The type is one of the simple types defined above, as denoted by the appropriate IDL reserved word. In an actual interface definition, the appropriate IDL reserved word is used to indicate the type. Exception values can also be sequences. Sequences are described below.

*Name* is the symbolic name of the exception value. The name of each value for an exception must be distinct for the exception.

15.1.4 Sequences

A sequence is a single-dimensional vector of sequential data values. Each data value is denoted by an index whose type is `long`. The values for these indices are sequential. The value of the first index is not specified; this value depends upon the interface implementation technology.

A sequence with no restrictions on the quantity of values it can contain is specified as:

```
datatype[] name
```

*Datatype* is the data type of the values in the sequence. The type is one of the simple types defined above, as denoted by the appropriate IDL reserved word. *Name* is the name of the property, input or output, or exception data value.
A sequence with restrictions on the quantity of values it can contain is specified as:

\[
datatype[\text{quantity}] \ name
\]

\textit{Quantity} is a numeric value that indicates the maximum quantity of values that the sequence can contain. A sequence may contain less than this quantity. The means by which the quantity of values in a sequence is determined depends upon the interface implementation technology.

### 15.1.5 Interface References

An interface reference enables access to a specific interface to a specific instance of a component that implements the interface. The interface reference data type represents an interface reference. The type of a property, method input, method output, and exception data value can be an interface reference:

\[
\text{interfacename name}
\]

\textit{Interfacename} is the name of the interface that the reference represents. \textit{Name} is the name of the property, input or output, or exception data value.

### 15.1.6 Principal Interface

The reserved word \textbf{Principal} is the interface name for a component’s principal interface. The role of a component’s principal interface is discussed in Section 6.1 \textbf{Component and Interface Concepts}. The type of a property, method input, method output, and exception data value can be an interface reference to a principal interface:

\[
\text{Principal name}
\]

\textit{Name} is the name of the property, input or output, or exception data value.

### 15.1.7 Qualifying Names

In the IDL there is never a case in which the names of properties, methods, and exceptions defined in one lexical scope are referenced in another lexical scope. However, when documenting the interfaces it can be useful to indicate the scope within which a particular property, method, or exception name has been defined.

The convention for doing so is to formulate a qualified name comprised of the name of the interface within whose scope the property, method, or exception of interest was defined, followed by a pair of colons (::) followed by the name of the property, method, or exception, for example:
1 ContextManager::JoinCommonContext
2
3 denotes the method JoinCommonContext as defined for the interface ContextManager.

15.2 Interface Implementation Issues

This section describes requirements that all CMA interface implementations must respect.

15.2.1 NotImplemented Exception

In the event that a method is not implemented, the exception NotImplemented shall be raised.
This exception can be raised, for example, when a method has been deprecated and is no
longer implemented by a CMA component. This exception can implicitly be raised by any
method defined using CMA IDL and need not be explicitly declared.

15.2.2 GeneralFailure Exception

In the event that a method cannot be properly performed due to an error or failure condition,
and an explicitly defined exception does not appropriate represent the situation, then the
exception GeneralFailure shall be raised. This exception might be raised, for example, when
a CMA component is unable complete a computation due to an internal error. This exception
can implicitly be raised by any method defined using CMA IDL and need not be explicitly
declared.

15.2.3 Coupon Representation

A participant coupon is a 32-bit integer, represented as the CMA IDL data type long, that is
assigned by a common context manager to denote each application that joins a common
context system. An application is assigned a participant coupon when it joins a common
context system. It subsequently uses the coupon to identify itself when performing methods
on the context manager.

A context coupon is a 32-bit integer that is assigned by a common context manager to denote
each context change transaction. Each time a new transaction is started a new coupon is
assigned by the context manager to denote the transaction. Applications use a context coupon
to denote the transaction of interest.

Participant coupons shall have unique values for the duration of a common context session
(i.e., from the time the first application joins to the time the last application leaves). Context
coupons shall also have unique values for the duration of a common context system.

The distinguished value of 0 shall never be assigned as a participant coupon value or as a
coupon value.
15.2.4 Format for Application Names

Several interfaces require that an application provide a CMA IDL string that contains a symbolic name for the application. This string is generally used to distinguish one application from another.

This string shall only be comprised of alphanumerical characters, blank spaces (no tabs), the underscore (_) and period (.) characters. The string shall neither begin nor end with a blank space.

Additionally, an application that is capable of allowing multiple instances of itself to execute on the same desktop shall append to the end of its symbolic name the number-score character (#) followed by a string that distinguishes one instance of the application from another.

The composition of the appended string is not specified, as long as no two running instances of the application running on a particular desktop use the same appended string at the same time. The appended string shall only be comprised of alphanumerical characters, blank spaces (no tabs), as well as the underscore (_) and period (.) characters. The appended string shall neither begin nor end with a blank space.

Character case is not considered when comparing application names.

The BNF for an application name appears below:

```
SPACE = " "
UNDERSCORE = "_"
PERIOD = "."
DASH = "-"
ALPHA = "a-zA-Z"
NUMERIC = "0-9"
POUND = "#"
```

```
initialAppChar = ALPHA | NUMERIC | UNDERSCORE | PERIOD
endAppChar = initialAppChar
appChar = initialAppChar | SPACE
appNameBody = appChar* endAppChar
appQualifier = POUND initialAppChar [appNameBody]
appName = initialAppChar [appNameBody] [appQualifier]
```

An example of an application name is:

```
"3M Clinical Workstation#0"
"3M Clinical Workstation#1"
"3M Clinical Workstation#2"
```

Application names formed as such shall be interpreted as representing the same logical application (e.g., “3M Clinical Workstation”) while also representing distinct running instances of the application (i.e., three instances of “3M Clinical Workstation”).
15.2.5 Extraneous Context Items

Context participants shall robustly deal with the situation in which context data items that they do not recognize are nevertheless part of the common context. This might occur, for example, in a system comprised of context participants that have been implemented using different versions of the CMA data definition specifications. A participant implemented using an earlier version of these specifications might not recognize context items defined in subsequent versions of the specifications. Context participants shall simply ignore context data items whose names they do not recognize.

Similarly, context managers shall allow any context data item for any CMA-defined subject to be part of the context, as long as the name for the item is properly formatted.

15.2.6 Forcing the Termination of a Context Change Transaction

The context manager may need to force the termination of a context change transaction when it appears that the instigator of the transaction has failed before completing the transaction. Specifically, it is recommended that any context manager method that can result in the ContextManager::TransactionInProgress exception being thrown should first explicitly confirm that the transaction instigator is still alive.

Most context manager implementations will employ a timer to monitor the activity of a transaction instigator. If the instigator does not perform the necessary operations on the context manager’s interfaces in a timely manner, it can be inferred that the instigator has failed. The method ContextParticipant::Ping is defined to enable the context manager to probe a context participant to determine its liveness. The context manager may additionally confirm the liveness of a context participant using technology-specific mechanisms.

The duration of these timers, and the use of confirmation techniques, are implementation-dependent.

The context manager shall clean up after the failure of the instigator by performing the following actions:

1. The coupon assigned by the manager for the transaction is invalidated.
2. The transaction-specific version of the context data is discarded.
3. The coupon and context data associated with the most recently committed transaction are unaffected.
4. The context manager’s internal state is set to indicate that there is no longer a transaction in progress.

Additional actions depend upon when the context manager determines that the instigator has failed, as described in Table 3: Handling Transaction Instigator Failure.
Instigator fails … | Leaving systems in the following state … | Context manager cleans-up by …
---|---|---
before ending the transaction (see ContextManager::EndContext Changes) | a context change transaction is in progress, although surveying has not yet been performed | performing the actions described above
after ending the transaction but before publishing its decision to accept or cancel the changes (see ContextManager::Publish ChangesDecision) | a context change is in progress and the surveyed participants are waiting for the survey decision | publishing the fact that the context changes have been canceled and then performing the actions described above

Table 3: Handling Transaction Instigator Failure

15.2.7 Character-Encoded Binary Data

Several of the CMA component interfaces use CMA IDL *string* parameters that contain character-encoded binary data. The following representation of character-encoded binary data shall be applied for all such parameters:

Each byte of data shall be represented by two printable characters. The four high bits of the byte (i.e., the high octet) shall be represented by the left character. The four low bits of the byte (i.e., the low octet) shall be represented by the right character.

An array of bytes shall be represented by character-encodings such that the left most character-encoded byte in the string represents the data byte at lowest array index. The encoding follows sequentially, such that the right most character-encoded byte in the string represents the data byte at the highest array index.

---

11 Base64 encoding was not selected as a character-encoding scheme for binary data, as the added compression offered by the scheme is of minimal advantage for the CMA, wherein only relatively small quantities of binary data are transmitted.
Each four bits of data (i.e., an octet) is represented by an alphanumeric character as follows:

<table>
<thead>
<tr>
<th>Data (Octet)</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A or a</td>
</tr>
<tr>
<td>1011</td>
<td>B or b</td>
</tr>
<tr>
<td>1100</td>
<td>C or c</td>
</tr>
<tr>
<td>1101</td>
<td>D or d</td>
</tr>
<tr>
<td>1110</td>
<td>E or e</td>
</tr>
<tr>
<td>1111</td>
<td>F or f</td>
</tr>
</tbody>
</table>

The actual character set that is employed is technology-specific. Each of the HL7 context management technology mapping specification documents indicates the character set that is used for a particular technology-specific implementation.

Binary data that is character-encoded as a string shall not include white space or any other characters other than the ones shown in the table above. The character-encoded string is not case sensitive. An example of binary data character-encoded per these conventions is:

<table>
<thead>
<tr>
<th>Binary Data:</th>
<th>Character-Encoded String:</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001 11101001 11000111 1000010</td>
<td>01E9C782</td>
</tr>
</tbody>
</table>

### 15.2.8 Representing Message Authentication Codes, Signatures and Public Keys

Message authentication codes, digital signatures, public keys are used as input or output parameters for several of the methods defined for CMA component interfaces. The CMA IDL data type for each of these parameters is `string`. Each string contains character-encoded binary data, encoded per Section [5.2.7][Character-Encoded Binary Data]

The binary data that is encoded is technology-specific. Each of the HL7 context management technology mapping specification documents indicates the binary data types needed for a particular technology-specific implementation. It is necessary that both the sender and receiver of a message authentication code, digital signature, or public key agree upon the
format of the underlying binary data type, and the algorithms used to create the data. The
method SecureBinding::InitializeBinding, defined in 15.3.9.1, enables this agreement to be
established.

15.2.9 Representing Basic Data Types as Strings
Several of the CMA component interfaces use input or output parameters whose values are
computed from the string representations of data values of various types. For example,
digital signatures are computed from a one-way hash value, which is, in turn, computed from
a string formed by concatenating a list of data values, each of which is represented as a
string.

The following data types shall be represented as character strings using the formats described
in Table 4: Character Representations for Basic Data Types.

<table>
<thead>
<tr>
<th>Type</th>
<th>String Representation</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>0, if false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1, if true</td>
<td></td>
</tr>
<tr>
<td>short</td>
<td>dddd, where d is a numeric character representing a decimal digit and the number of characters depends upon the value of the number.</td>
<td>Leading minus sign (-dddd) if number is negative. No plus sign if positive.</td>
</tr>
<tr>
<td>long</td>
<td>Same as for short.</td>
<td></td>
</tr>
<tr>
<td>date</td>
<td>yyyy/mm/dd hh:mm:ss</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>As is.</td>
<td>Case is preserved.</td>
</tr>
<tr>
<td>float</td>
<td>dddd.dddd, where d is a numeric character representing a decimal digit. The number of digits before the decimal point depends on the magnitude of the number. The number of digits after the decimal point shall not exceed four, and other than the first digit after the decimal trailing zero’s shall not be used.</td>
<td>Leading minus sign (-dddd.dddd) if number is negative. No plus sign if positive.</td>
</tr>
<tr>
<td>double</td>
<td>Same as float, except that the number of digits after the decimal shall not exceed eight.</td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>bb, where b is a hexadecimal digit. The byte is represented as unsigned.</td>
<td>Lower case for alphabetic characters that represent hex digits (i.e., a, b, c, d, e, f).</td>
</tr>
</tbody>
</table>

Table 4: Character Representations for Basic Data Types
The actual character set that is employed is technology-specific. Each of the HL7 context management technology mapping specification documents indicates the character set that is used for a particular technology-specific implementation.

15.2.10 Pre-Defined Context Agent Coupons
A participant coupon value is pre-defined for each type of context agent. In general, a negative coupon value denotes a context agent, as opposed to a context participant application. The following values are currently allocated:

<table>
<thead>
<tr>
<th>Context Agent</th>
<th>Coupon Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient Mapping Agent</td>
<td>-1</td>
</tr>
<tr>
<td>User Mapping Agent</td>
<td>-2</td>
</tr>
<tr>
<td>Encounter Mapping Agent</td>
<td>-3</td>
</tr>
<tr>
<td>Observation Mapping Agent</td>
<td>-4</td>
</tr>
<tr>
<td>Certificate Annotation Agent</td>
<td>-5</td>
</tr>
<tr>
<td>Reserved for future</td>
<td>-6 through –9,999</td>
</tr>
<tr>
<td>Reserved for Context Agents for Custom Subjects</td>
<td>-10,000 through –20,000</td>
</tr>
</tbody>
</table>

Pre-defined coupon values are used for context agents because they do not explicitly join the context system. Instead, a context agent is implicitly “pulled” into the context system each time a context change transaction occurs, when the context manager performs the method MappingAgent::ContextChangesPending on a mapping agent, or the method ContextAgent::ContextChangesPending on a mapping agent or annotation agent. See Sections 15.3.8 MappingAgent (MA) and 15.3.3 ContextAgent (CA).

However, context agents such as the user mapping agent need to know their participant coupon values prior to the first context change transaction. For example, the user mapping agent needs to establish a secure binding with the context manager before it can set user context items. In order to establish this binding, the user mapping agent must present the context manager with its coupon. By having a priori knowledge of its coupon value, the user mapping agent can establish its secure binding whenever it decides to, up until the time it actually attempts to set the context.

Context agents for standard subjects are assigned coupon values, as indicated in the table above. Any implementation of a context agent for a standard subject shall use the coupon value assigned for such a context agent. Any implementation of a context manager shall use the assigned coupon value for a context agent for a standard subject.
A context agent for a custom subject shall use a coupon value that is assigned on site at the
time at which the context agent is deployed. This coupon shall be within the range of values
indicated in the table above. Each site shall ensure that the coupon value that it assigns to a
custom context agents deployed at the site is different for each type of custom context agent
deployed at the site.

Both a custom context agent and the context manager shall provide a means for being
informed on site as to which coupon value is to be used. A technology-specific means for
facilitating this process may be specified in each of the HL7 context management technology
mapping specification documents.
15.3 **Interfaces**

This section specifies the methods for each of the CMA interfaces.

### 15.3.1 AuthenticationRepository (AR)

```java
interface AuthenticationRepository {
    exception AuthenticationFailed { string reason; }
    exception UnknownApplication {}
    exception UnknownConnection {}
    exception LogonNotFound { string logonName; }
    exception UnknownDataFormat { string dataFormat; }

    Connect
    inputs(string applicationName)
    outputs(long connectionCoupon)
    raises(UnknownApplication)

    Disconnect
    inputs(long connectionCoupon)
    outputs()
    raises(UnknownConnection)

    SetAuthenticationData
    inputs(long connectionCoupon, string logonName, string dataFormat,
    string userData, string appSignature)
    outputs()
    raises(UnknownConnection, AuthenticationFailed)

    DeleteAuthenticationData
    inputs(long connectionCoupon, string logonName, string dataFormat,
    string appSignature)
    outputs()
    raises(UnknownConnection, AuthenticationFailed, LogonNotFound,
    UnknownDataFormat)

    GetAuthenticationData
    inputs(long connectionCoupon, string logonName, string dataFormat,
    string appSignature)
    outputs(string userData, string repositorySignature)
    raises(UnknownConnection, AuthenticationFailed, LogonNotFound,
    UnknownDataFormat)
}
```

### 15.3.2 Synopsis

This interface enables a context participant to securely maintain the data it uses to authenticate its users in an external repository.
15.3.2.1 Connect

This method enables an application to establish a connection with the authentication repository. An application must have a connection before it can set or get user authentication data.

The value of the input applicationName is a succinct string that contains the application’s symbolic name. The output connectionCoupon is the value of a connection coupon that the application can subsequently use to denote itself when performing other authentication repository methods.

The value of input applicationName is used by the authentication repository to determine the passcode for an application. The passcode is needed when an application establishes a secure binding with the authentication repository (see Section 15.3.9 SecureBinding (SB)). Multiple instances of an application can connect to the authentication repository using the same name. Each instance of the application will be assigned a unique connection coupon. Each instance of the application will need to establish a secure binding with the repository.

The value of the input applicationName is also used by the authentication repository to store/retrieve the user authentication data within the repository.

The exception UnknownApplication is raised if the input applicationName does not represent an application currently known to the authentication repository.

15.3.2.2 Disconnect

This method enables an application to disconnect from the authentication repository. An application shall disconnect before it terminates. The value of the input connectionCoupon denotes the application.

The exception UnknownConnection is raised if the input connectionCoupon does not denote an application currently connected to the authentication repository.

15.3.2.3 SetAuthenticationData

This method enables an application to store authentication data for a particular user’s logon name within the authentication repository. This method also enables an application to update authentication data for a particular user’s logon name that it has already stored in the repository.

The value of the input connectionCoupon denotes the application, the value of the input logonName is a user’s logon name, the value of the input userData is the application-specific data used to authenticate the user, and the value of the input appSignature is the application’s digital signature. This signature enables the authentication repository to authenticate that the request to set the authentication data came from the application denoted
by the value of connectionCoupon, and that the values of connectionCoupon, logonName, dataFormat, and userData, were not tampered with between the time they were sent and were received.

Concatenating the string representations of the following inputs in the order listed shall form the data from which a message digest is computed by the application:

- connectionCoupon
- logonName
- dataFormat
- userData

An application shall compute its digital signature by encrypting the message digest with its private key.

The value of the input dataFormat is an application-defined string that is used when an application needs to maintain multiple forms of authentication data for a user (e.g., password, thumbprint image, etc.). If only one form of authentication data is needed, this string can be empty (""). Multiple calls of SetAuthenticationData are required to set different forms of authentication data for a particular user. The value of dataFormat for each call shall indicate the form of authentication data to be stored.

The value of the input userData contains user authentication data that has been encrypted by the application using an encryption technique chosen by the application. This data is character-encoded per Section 15.2.7, Character-Encoded Binary Data. The structure of the encoded binary data is application-dependent and is not specified.

The exception UnknownConnection is raised if the input connectionCoupon does not denote an application that is currently connected to the repository.

The exception AuthenticationFailed is raised if the process of authentication determines that the signature is not the signature for the application denoted by the input connectionCoupon or that the input parameter’s values have been tampered with.

### 15.3.2.4 DeleteAuthenticationData

This method enables an application to delete from the authentication repository some or all of the authentication data that it previously stored for a particular logon name. Both the logon name and the associated authentication data are deleted.

The value of the input connectionCoupon denotes the application and the value of the input logonName is the logon name to be deleted.
The value of the input `dataFormat` is an application-defined string that is used when an
application maintains multiple forms of authentication data for a user (e.g., password,
thumbprint image, etc.) within the repository. If this string is empty, then all of the forms of
authentication data stored for the user are deleted. If this string is not empty, then just the
denoted form of authentication data is deleted.

The value of the input `appSignature` is the application’s digital signature.

Concatenating the string representations of the following inputs in the order listed shall form
the data from which a message digest is computed by the application:

- `connectionCoupon`
- `logonName`
- `dataFormat`

An application shall compute its digital signature by encrypting the message digest with its
private key.

This signature enables the authentication repository to authenticate that the request to delete
the authentication data came from the application denoted by the value of
`connectionCoupon`, and that the values of `coupon`, `logonName`, and `dataFormat` were not
tampered with between the time they were sent and were received.

The exception `UnknownConnection` is raised if the input `connectionCoupon` does not denote
an application that is currently connected to the repository.

The exception `AuthenticationFailed` is raised if the process of authentication determines that
the signature is not the signature for the application denoted by the input `connectionCoupon`
or that the input parameter values have been tampered with.

The exception `LogonNotFound` is raised if user authentication data corresponding to the
logon name denoted by the input `logonName` does not reside in the repository.

The exception `UnknownDataFormat` is raised if the form of authentication data denoted by
the input `dataFormat` is not found in the repository.

### 15.3.2.5 GetAuthenticationData

This method enables an application to retrieve from the authentication repository the
authentication data previously stored for a particular user’s logon name. The value of the
input `connectionCoupon` denotes the application, the value of the input `logonName` is a user’s
logon name, and the value of the input `appSignature` is the application’s digital signature.
This signature enables the authentication repository to authenticate that the request to get the authentication data came from the application denoted by the value of connectionCoupon, and that the values of coupon, logonName, and dataFormat were not tampered with between the time they were sent and were received.

Concatenating the string representations of the following inputs in the order listed shall form the data from which a message digest is computed by the application:

- connectionCoupon
- logonName
- dataFormat

An application shall compute its digital signature by encrypting the message digest with its private key.

The value of the input dataFormat is an application-defined string that is used when an application needs to maintain multiple forms of authentication data for a user (e.g., password, thumb-print image, etc.). If only one form of data is used, this string can be empty. Multiple calls of GetAuthenticationData are required to get different forms of authentication data for a particular user. The value of dataFormat for each call shall indicate the form of authentication data to be retrieved.

The value of the output userData is the application-specific data used to authenticate the user. The output userData remains encrypted, as it was when it was stored by the application using SetAuthenticationData.

The output userData shall be used as the data from which a message digest is computed by the application. The authentication repository shall compute its digital signature by encrypting the message digest with its private key.

This signature enables the application to authenticate that the authentication data returned by this method came from the authentication repository and that the value of userData was not tampered with between the time it was sent and was received.

The exception UnknownConnection is raised if the input connectionCoupon does not denote an application that is currently connected to the repository.

The exception AuthenticationFailed is raised if the process of authentication determines that the signature is not the signature for the application denoted by the input connectionCoupon or that the input parameter values have been tampered with.

The exception LogonNotFound is raised if user authentication data corresponding to the logon name denoted by the input logonName does not reside in the repository.
The exception UnknownDataFormat is raised if the form of authentication data denoted by the input `dataFormat` is not found in the repository.
15.3.3 ContextAgent (CA)

interface ContextAgent {
  ContextChangesPending
  inputs(long agentCoupon, Principal contextMgr,
         string[] itemNames, string[] itemValues, long contextCoupon,
         string managerSignature)
  outputs(long agentCoupon, string[] itemNames, string[] itemValues,
          long contextCoupon, string agentSignature,
          string decision, string reason)
  raises()

  Ping
  inputs()
  outputs()
  raises()
}

15.3.3.1 Synopsis
This interface enables a context agent to be notified about context changes. A context agent that serves as a mapping agent may implement this interface as an alternative to the interface MappingAgent (See Section [5.3.8]).

15.3.3.2 ContextChangesPending
This method informs a context agent (e.g., mapping agent, annotation agent) in a common context system that a change to the common context data is pending.

15.3.3.2.1 Inputs
The value of the input contextCoupon denotes the transaction within which the context changes occurred. The value of the input agentCoupon is a predefined coupon that denotes the specific type of mapping agent. (See Section [5.2.10] Pre-Defined Context Agent Coupons). The value of the input contextMgr is an interface reference to the context manager’s principal interface. This is so that the context agent can easily obtain the context manager interface(s) it needs.

As an optimization, the context manager provides the context agent with a relevant set of context data items. In most cases, it should be unnecessary for an agent to obtain additional data from the context manager. However, the context manager methods ContextData::GetItemValues and/or SecureContextData::GetItemValues may still be used by a context agent.

When the agent is a mapping agent, the context data shall include all of the items for the context subject whose identifier items the agent is supposed to map. For example, for a
patient mapping agent, these items will contain all of the items currently set for the patient subject.

When the agent is an annotation agent, the context data shall include all of the items for the subject upon which the annotation subject depends.

The names of the context items are contained in the input sequence itemNames. The values for each of these items are contained in the input sequence itemValues. The i\textsuperscript{th} element in itemValues is the value for the item named by the i\textsuperscript{th} element in itemNames. Note that the representation of an item value is always a string, character-encoded as necessary per the conventions defined in Section 5.2 Interface Implementation Issues.

If the context items contained in the inputs itemNames and itemValues are for a secure subject, then the context manager’s digital signature shall be provided as the value of the input managerSignature. This signature enables the agent to authenticate that the context items came from the real context manager, and that the values were not tampered with between the time they were sent and were received.

Concatenating the string representations of the following inputs in the order listed shall form the data from which a message digest is computed by the context manager:

- agentCoupon
- itemNames (i.e., All the elements in the order that they appear in the array.)
- iItemValues (i.e., All the elements in the order that they appear in the array.)
- contextCoupon

The context manager shall compute its digital signature by encrypting the message digest with its private key.

If the context items contained in the inputs itemNames and itemValues are not for a secure subject, then the value of the input managerSignature shall be an empty string (""").

15.3.3.2.2 Outputs

As an optimization, the context agent may provide the context manager with the set of context data items that should be added to the context. (Note that a context agent may only add data to the context.) In most cases, it should be unnecessary for an agent to need to set additional data via the context manager’s ContextData or SecureContextData interfaces. However, the context manager methods ContextData::SetItemValues and/or SecureContextData::SetItemValues may still be used by a context agent.
When used by a context agent, the outputs itemNames and itemValues contain the names and values for the context items whose values are to be set by the context manager. The names of the context items are contained in the output sequence itemNames. The values for each of these items are contained in the output sequence itemValues. The ith element in itemValues is the value for the item named by the ith element in itemNames. Note that the representation of an item value is always a string, character-encoded as necessary per the conventions defined in Section 15.2 Interface Implementation Issues.

When the context items to be set belong to a secure subject, then the agent shall also provide its digital as the value of the output agentSignature. This signature enables the context manager to authenticate that they came from a valid secure subject context agent, and that the values were not tampered with between the time they were sent and were received.

Concatenating the string representations of the following outputs in the order listed shall form the data from which a message digest is computed by the context agent:

- agentCoupon
- itemNames (i.e., All the elements in the order that they appear in the array.)
- itemValues (i.e., All the elements in the order that they appear in the array.)
- contextCoupon

An agent shall compute its digital signature by encrypting the message digest with its private key.

If the context items contained in the outputs itemNames and itemValues are not for a secure subject, then the value of the output agentSignature shall be an empty string (“”).

15.3.3.2.3 Outputs for Mapping Agents

If the agent is not a mapping agent, then the outputs decision and reason are not used. The value of the output decision and the value of the output reason shall be an empty string (“”).

Otherwise, if the agent is a mapping agent, then these outputs are used. A mapping agent shall respond with an indication of how it wants to deal with the context change:

- The changes are valid
- The changes are invalid

If the changes are valid, then the value of the output decision shall be “valid”. If the changes are invalid, then the value of the output decision shall be “invalid”. The changes shall only be declared invalid if the set of identifiers in the proposed context data do not all represent the same real-world entity or concept. If the changes are invalid, then the value of the output
reason shall contain a succinct but detailed string describing why the changes were invalid. Otherwise the value of reason shall be an empty string ("").

The context manager shall treat the value of the output decision in a case-insensitive manner. The context manager shall preserve the case used in the value of the output reason.

15.3.3.2.4 Exceptions
This method does not raise any exceptions.

However, there are nevertheless a number of incorrect behaviors that are possible concerning the values of the outputs. When a context manager detects any of the following incorrect behaviors, it shall not update the context data with the context items provided by the agent as the values of the outputs itemNames and itemValues:

- The value of agentSignature is not a digital signature and a signature is required in order to perform this method.

- A value for agentSignature is required and provided, but the process of authentication determines that: the agent did not previously provide its public key via the context manager’s interface SecureBinding; that the input agentSignature has been forged; that the input parameter values have been tampered with; that the agent has not been designated for setting secure context data.

- The output agentCoupon does not denote a context agent that is currently a participant in the common context system.

- The output contextCoupon does not denote the transaction currently in progress.

- The number of items in the output nitemNames does not match the number of items in the output itemValues.

- The format of an item named in the output itemNames for deletion does not conform to the specification for the item in the relevant HL7 Context Management Data Definition Specification.

- The data type for one or more of the items whose value is contained in the output itemValues is not the same as the expected data type.

- The data value for one or more of the items whose value is contained in the output itemValues is determined to be unacceptable. This exception is used by context manager implementations that enforce semantic constraints on the common context. Not all context manager implementations will do this.
• The value for one or more of the items named in the output itemNames has already been set by the application that instigated the context change transaction.

• The value for one or more of the items named in the output itemNames is for a context subject other than the subject that the agent is for (e.g., the user certificate annotation agent returns context item values for the patient identity subject).

The context manager implementation may opt to log or otherwise record these errors. However, the means for doing so is beyond the scope of this specification. Other than through the explicit intervention of a systems administrator, a context agent implementation may not necessarily be aware that any of these errors have transpired.

15.3.3.3 Ping

This method provides a means for a context manager to determine whether or not a context agent in a common context system is still running. This method shall be implemented by all agents to return immediately. The context manager can then perform this method to probe a context agent when the agent’s existence is in doubt.

In performing this method, the context manager will be able to indirectly exercise the underlying communications infrastructure. The infrastructure will either indicate that the method was successfully performed, that the method failed because the agent no longer exists, or that the method failed but it cannot be determined whether or not the agent exists. In this last case, the manager shall assume that the agent still exists.
15.3.4 ContextData (CD)

interface ContextData {
  exception UnknownParticipant { long participantCoupon; }
  exception UnknownItemName { string itemName; }
  exception BaditemNameFormat { string itemName; string reason }
  exception BadItemType { string itemName; type actual;
    type expected; }
  exception BadItemValue { string itemName; variant itemValue;
    string reason; }
  exception NameValueCountMismatch {long numNames; long numValues }
  exception ChangesNotPossible {}
  exception ChangesNotAllowed {}
  exception InvalidContextCoupon {}
}

GetItemNames
inputs(long contextCoupon)
outputs(string[] names)
raises(InvalidContextCoupon)

DeleteItems
inputs(long participantCoupon, string[] itemNames,
  long contextCoupon)
outputs()
raises(NotInTransaction, UnknownParticipant, InvalidContextCoupon,
  BaditemNameFormat, UnknownItemName, ChangesNotPossible,
  ChangesNotAllowed)

SetItemValues
inputs(long participantCoupon, string[] itemNames,
  variant[] itemValues, long contextCoupon)
outputs()
raises(NotInTransaction, UnknownParticipant, InvalidContextCoupon,
  NameValueCountMismatch, BaditemNameFormat, BadItemType,
  BadItemValue, ChangesNotPossible, ChangesNotAllowed)

GetItemValues
inputs(string[] itemNames, boolean onlyChanges, long contextCoupon)
outputs(variant[] itemValues)
raises(InvalidContextCoupon, BaditemNameFormat, UnknownItemName)
}

15.3.4.1 Synopsis

This interface enables a context participant to get and set context data for common subjects, which are subjects for which secure access is not required. The data is represented as a set of items, each of which is structured as a name/value pair.
15.3.4.2 GetItemNames

This method enables a participant in a common context system to obtain the names of the common context items.

This method can be performed outside the scope of a context change transaction. In this case, the value of the input contextCoupon must denote the most recently committed transaction. The output itemNames is a sequence containing the item names that represent the state of the common context as it was when the most recently committed transaction was completed.

This method can also be performed within the scope of a context change transaction that is currently in progress. In this case, the input contextCoupon must denote the current transaction. The output itemNames contains the item names that represent the state of the common context as it has been established so far by the transaction. The output itemNames is empty (i.e. zero elements) until a participant explicitly sets item values via the ContextData::SetItemValues method within the scope of the transaction.

The exception InvalidContextCoupon is raised if the input contextCoupon does not denote the most recently committed transaction or the transaction currently in progress.

15.3.4.3 DeleteItems

Note: This method has been deemed extraneous and is being deprecated. In a future version of this specification context managers may chose to not implement this method even though it remains part of the ContextData interface definition.

This method enables an application in a common context system to remove an item from the set of common context items. The application or mapping agent denotes itself with its participant coupon as the value of the input participantCoupon. The value of the input contextCoupon must denote the current context change transaction, as obtained by the instigator of the transaction when it performed the ContextManager::StartContextChanges method.

The exception NotInTransaction is raised if there is no change transaction currently in progress.

The exception UnknownParticipant is raised if the input participantCoupon does not denote an application or mapping agent that is currently a participant in the common context system.

The exception InvalidContextCoupon is raised if the context coupon parameter does not denote the transaction currently in progress.

The exception BadItemNameFormat is raised if the format of an item named for deletion does not conform to the specification for the item in the relevant HL7 context management data definition specification document.
The exception UnknownItemName is raised if one or more of the items named for deletion is not the name of an item in the context as it stands under the current transaction.

The exception ChangesNotPossible is raised if the ContextData::DeleteItems method is invoked after the ContextManager::EndContextChanges method has already been invoked for the transaction currently in progress.

The exception ChangesNotAllowed is raised by ContextData::DeleteItems if a mapping agent attempts to delete context items, or attempts to set context data items that have already been set, or if any participant attempts to set annotating data items and the participant has not been designated as an authentic source for these data items.

### 15.3.4.4 SetItemValues

This method enables an application or mapping agent in a common context system to set the value of one or more common context items. The application or mapping agent denotes itself with its participant coupon as the value of the input `participantCoupon`. The names of the context items to be set are contained in the input sequence `itemNames`. The values for each of these items are contained in the input sequence `itemValues`. The i\(^{th}\) element in `itemValues` is the value for the item named by the i\(^{th}\) element in `itemNames`.

If an item named in `itemNames` is not currently an item in the common context, it will be added. The data type for a newly added item is the same as the data type of the element in `itemValues` that contains the item’s value.

This method can only be performed within the scope of a context change transaction. The value of the input `contextCoupon` must denote the current transaction.

The exception NotInTransaction is raised if there is no change transaction currently in progress.

The exception UnknownParticipant is raised if the input `participantCoupon` does not denote an application or mapping agent that is currently a participant in the common context system.

The exception InvalidContextCoupon is raised if the input `contextCoupon` does not denote the transaction currently in progress.

The exception NameValueCountMismatch is raised if the number of items in the input `nitemNames` does not match the number of items in the input `itemValues`.

The exception BadItemNameFormat is raised if the format of an item named for deletion does not conform to the specification for the item in the relevant HL7 Context Management Data Definition Specification.
The exception BadItemType is raised if the data type for one or more of the items whose value is to be set is not the same as the expected data type.

The exception BadItemValue is raised if the data value for one or more of the items whose value is to be set is determined to be unacceptable. This exception is used by context manager implementations that enforce semantic constraints on the common context. Not all context manager implementations will do this.

The exception ChangesNotPossible is raised if the ContextData::SetItemValues method is invoked by an application after the ContextManager::EndContextChanges method has already been invoked for the transaction currently in progress. (This exception is not raised if a mapping agent invokes ContextData::SetItemValues after ContextManager.)

The exception ChangesNotAllowed is raised if the caller is a context agent that has attempted to set a value for a context subject other than the subject that it is the agent for. This exception is also raised if the caller is a context agent that has attempted to set a value for a context item for which a value has already been set by the application that instigated the context change transaction. Finally, this exception is raised if the caller has attempted to set a value for an annotation subject and the caller is not an annotation agent.

15.3.4.5 GetItemValues

This method enables a participant in a common context system to obtain the value of one or more context items.

When the value of the input contextCoupon denotes the most recently committed transaction, the item values that are returned represent the state of the common context as it existed when the transaction was completed. This is true even if there is currently a new transaction in progress.

When the value of the input contextCoupon denotes the transaction currently in progress, the item values that are returned represent the state of the common context as it has been established so far by the transaction. The capability to access the items for the transaction in progress enables applications to use this information to determine how they want to respond to the context change survey conducted by the context manager (see Sections 8.8 Context Change Notification Process and 15.3.6 ContextParticipant (CP)). For example, an imaging application that caches data may respond to the survey differently depending upon whether the proposed context change involves a patient currently in the application’s cache.

The items of interest are indicated in the input sequence itemNames. These names can be fully-qualified item names, which means that the all of the fields for an item’s name are explicitly specified (e.g., "Patient.Id.MRN.St_Elsewhere_Hospital").
Alternatively, a wild card represented by an asterisk (*) can be used in place of a specific string for any of the item name fields except for the subject field (which is lexically the first field on the left). The wild card enables a participant to obtain one or more items without having to specify complete item names.

If a wild card is used, it must appear in only the last field specified in the item name string (which is lexically the last field on the right). Additional field names and/or wild cards must not appear after a wild card (i.e., lexically to the right of the wild card). Examples of properly formatted item names include:

- “Patient.*” matches all of the context items for the patient subject
- “Patient.Id.*” matches all of the patient identifier items
- “Patient.Id.MRN.*” matches all of the patient identifiers that are site-specific medical record numbers

Conversely, “Patient.Id.*.*” and “Patient.Id.*.St_Elsewhere_Hospital” are examples of improperly formatted item names.

The sequence output itemValues contains the values of all of the items whose names match the set of names specified in the input itemNames. A specific item’s value will be included at most once in itemValues, even if its name matches more than one of the names specified in itemNames. For example, even if itemNames includes the names:

- “Patient.Id.MRN.St_Elsewhere_Hospital”
- “Patient.Id.*”

the value for the item named “Patient.Id.MRN.St_Elsewhere_Hospital” will be included only once in itemValues.

The elements in the sequence itemValues alternate between the complete name of an item (represented as a string) and the corresponding item value (represented by the appropriate data type). For example, if several context data items are returned, then the first element in the list is the name of the first item, the second element in the list is the value of the first item, the third element in the list is the name of the second item, the fourth element in the list is the value of the second item, and so on.

The input onlyChanges enables a participant to instruct the context manager to filter which items it returns no matter what names were specified. When the value of onlyChanges is true, then the items that are returned are limited to only the context subjects whose items were set
by the most recently committed context change transaction, or by the transaction in progress, as indicated by the value of \textit{contextCoupon}.

For example, if \textit{onlyChanges} is true, \textit{contextCoupon} denotes the most recently committed context change transaction, and \textit{itemNames} includes the name:

\begin{verbatim}
  "Patient.Id.*"
\end{verbatim}

but items in the patient subject were not set during the transaction, then the output \textit{itemValues} will not contain any items pertaining to the patient subject.

The exception \texttt{InvalidContextCoupon} is raised if the input \textit{contextCoupon} does not denote the most recently committed transaction or the transaction currently in progress.

The exception \texttt{BadItemNameFormat} is raised if the format of the name of one or more of the items whose value is to be set does not conform to the specification for the item in the relevant HL7 context management data definition specification.

The exception \texttt{UnknownItemName} is raised if one or more of the items named, wild cards notwithstanding, is not the name of an item in the context as it stands under the current transaction.
15.3.5 ContextManager (CM)

```java
interface ContextManager {
    exception AlreadyJoined {}
    exception UnknownParticipant { long participantCoupon; }
    exception TransactionInProgress { string instigatorName; }
    exception NotInTransaction {}
    exception InvalidTransaction { string reason; }
    exception TooManyParticipants { long howMany; }
    exception ChangesNotEnded {}
    exception AcceptNotPossible {}
    exception UndoNotPossible {}
    exception InvalidContextCoupon {}

    readonly long MostRecentContextCoupon

    JoinCommonContext
        inputs(ContextParticipant contextParticipant,
            string applicationName, boolean survey, boolean wait)
        outputs(long participantCoupon)
        raises(AlreadyJoined, TooManyParticipants, TransactionInProgress)

    LeaveCommonContext
        inputs(long participantCoupon)
        outputs()
        raises(UnknownParticipant)

    StartContextChanges
        inputs(long participantCoupon)
        outputs(long contextCoupon)
        raises(UnknownParticipant, TransactionInProgress, InvalidTransaction)

    EndContextChanges
        inputs(long contextCoupon)
        outputs(boolean noContinue, string[] responses)
        raises(InvalidContextCoupon, NotInTransaction, InvalidTransaction)

    UndoContextChanges
        inputs(long contextCoupon)
        outputs()
        raises(InvalidContextCoupon, NotInTransaction, UndoNotPossible)
```
PublishChangesDecision
inputs(long contextCoupon, string decision)
outputs()
raises(NotInTransaction, InvalidContextCoupon, ChangesNotEnded,
   AcceptNotPossible)

SuspendParticipation
inputs(long participantCoupon)
outputs()
raises(UnknownParticipant)

ResumeParticipation
inputs(long participantCoupon, boolean wait)
outputs()
raises(UnknownParticipant, TransactionInProgress)

15.3.5.1 Synopsis
This interface enables a context participant to join and leave a common context system, and
to perform context change transactions.

15.3.5.2 MostRecentContextCoupon
This read-only property contains the value of the context coupon that represents the most
recently committed changes to the common context data. Even if there is a change
transaction in progress, this property’s value represents the previously committed
transaction. If no transactions have been committed, the value of this property is 0.

15.3.5.3 JoinCommonContext
This method enables an application to join a common context system. The application must
provide a reference to its ContextParticipant interface as the value of the input
contextParticipant.

The value of the input applicationName is a succinct string that can be used to easily and
clearly identify the application to the user (see Section [5.2.4] Format for Application
Names). This string must be unique relative to the other applications that have already joined
the common context system.

If an application subsequently attempts to establish a secure binding with the context
manager (see Section [5.3.9] SecureBinding (SB)), then this string is used by the context
manager to determine the passcode for an application.

The application can also indicate whether it wants to participate in context change surveys
(the value of the input survey indicates true), or that it just wants to be informed when a
context change has been accepted (the value of the input survey indicates false).
An application can only join a common context system between context change transactions. If no transaction is in progress, the application is able to immediately join the context change system.

If a transaction is in progress and the value of the input `wait` indicates true, this method will block until the transaction completes. It is recommended that an application that is willing to wait also display a message to the user indicating that it is attempting to join a common context system. If a transaction is in progress and the value of the input `wait` indicates false, this method immediately raises the exception TransactionInProgress.

The output `participantCoupon` is the value of the participant coupon that the application can subsequently use to denote itself when performing other ContextManager methods.

The exception AlreadyJoined is raised if an application with the same name as the value of `applicationName` has already joined the context.

The exception TooManyParticipants is raised if the context manager is unable to accommodate an additional common context participant.

15.3.5.4 LeaveCommonContext

This method enables an application that is a participant in a common context system to leave the system. The application denotes itself using its participant coupon as the value of the input `participantCoupon`. Once this method returns, the application shall dispose of any context manager interface references it possesses and the application is free to terminate.

In order to avoid a deadlock condition, this method does not block. If this method was allowed to block, it would be possible for an application to block while the context manager was attempting to perform a method on the application’s ContextParticipant interface. For single-threaded applications, this could cause a deadlock.

Consequently, if a context change transaction is in progress when this method is called, the application may still be notified about the context change even though it has left the common context. The application is free to ignore this notification or may not even be capable of responding. The context manager will robustly handle the failure of an application to respond.

The exception UnknownParticipant is raised if the input `participantCoupon` does not denote an application that is currently a participant in the common context system.

15.3.5.5 StartContextChanges

This method enables an application to indicate that it wants to start changing the common context. The application denotes itself with its participant coupon as the value of the input `participantCoupon`. A context change transaction is initiated. Actual changes to the context
data are conducted via the ContextData interface. The output contextCoupon is the value of
the context coupon that has been assigned by the context manager to denote the change
transaction.

The context manager will automatically terminate context change transaction if it does not
detect activity on its ContextData interface or if the ContextManager::EndContextChanges
method is not performed in a timely manner. The amount of time that the manager will wait
before terminating the transaction depends upon the manager’s implementation.

The exception UnknownParticipant is raised if the input participantCoupon does not denote
an application that is currently a participant in the common context system.

The exception TransactionInProgress is raised if a context change transaction is already in
progress.

The exception InvalidTransaction is raised if a suspended application calls this method.

15.3.5.6 EndContextChanges

This method enables the application that instigated a context change transaction to indicate
that it has completed its changes to the common context. The value of the input
certaintyCoupon denotes the transaction currently in progress. This method initiates the two-
step change notification process and returns after the first phase of the notification process is
conducted by the context manager. During the first phase, the applications in the common
context system are surveyed to determine their ability or willingness to apply the context
changes. The ContextParticipant::ContextChangesPending method is performed on each
application in the survey.

The output responses is a sequence of strings that is used to convey the results of the survey
to the application that instigated a context change transaction.

If all of the applications surveyed indicate that they are willing to accept the context changes,
then the output sequence responses is empty (i.e. zero elements) and the output noContinue
is false. The sequence is empty because there is no useful information to be conveyed about
the applications that have accepted, other than the fact that they all accepted. The method
ContextManager::PublishChangesDecision with the decision accept shall be subsequently
performed by the instigating application to communicate to the other applications the
decision to accept the context changes and to complete the transaction.

If there are surveyed applications that either are unable to provide a response to the survey
(e.g., because they are “busy”), or that want to inform the user that work-in-progress might
be lost if the context is changed, then the return value contains a string for each such
application. The application that invoked this method is expected to display the strings to the
user and to obtain guidance about how to proceed.
The output `noContinue` indicates true if the mapping agent invalidated the transaction, or at least one of the surveyed applications is “busy”. It is not possible for the user to continue to apply the context change transaction if the value of `noContinue` is true. The only option the user has is to cancel the change or to disconnect the instigating application from the common context system. For either user decision, the method `ContextManager::PublishChangesDecision` with the decision `cancel` shall be performed by the instigating application.

If the mapping agent has not invalidated the transaction and there are no busy applications (i.e., `noContinue` is false), but there are applications that have conditionally accepted the context changes, the user can instruct the instigating application to apply the context changes anyway, cancel the changes, or to disconnect from the common context system.

The method `ContextManager::PublishChangesDecision` with the decision `accept` shall be subsequently performed by the instigating application to complete the transaction if the user decides to apply the context changes.

The method `ContextManager::PublishChangesDecision` with the decision `cancel` shall be subsequently performed by the instigating application to complete the transaction if the user decides to cancel the context changes or to disconnect the instigating application from the common context system.

The exception `InvalidContextCoupon` is raised if the input `contextCoupon` does not denote the transaction currently in progress.

The exception `NotInTransaction` is raised if there is no change transaction currently in progress.

The exception `InvalidTransaction` is raised if, for each subject whose context data items have been set by the application during the transaction the context data that has been set does not include at least one identity subject. This exception is also raised if the items set for each identity subject set by the application do not include at least one item that is a subject identifier item (e.g., context data for an identity subject cannot be comprised of just corroborating data). This exception is also raised if context data items have been set for multiple context subjects and a semantic dependency that pertains to two or more of these subjects has not been satisfied.

### 15.3.5.7 UndoContextChanges

This method enables an application to discard any context data changes that it has already made. The context coupon parameter denotes the transaction currently in progress. The current transaction is brought to a close and the context coupon is no longer valid.
The exception InvalidContextCoupon is raised if the input `contextCoupon` does not denote the transaction currently in progress.

The exception NotInTransaction is raised if there is no change transaction currently in progress.

The exception UndoNotPossible is raised if the method ContextManager::UndoContextChanges is attempted after the ContextManager::EndContextChanges method has been performed during the course of the current transaction.

### 15.3.5.8 PublishChangesDecision

This method enables the application that instigated a context change transaction to inform the other applications in a context system about whether the changes are to be applied or have been canceled. The value of the input `contextCoupon` denotes the transaction currently in progress.

The decision to accept the changes shall be published when the context changes are to be applied. The only times that context changes cannot be applied are when there were applications for which it was not possible to obtain a survey response (e.g., these applications were “busy”) or when a mapping agent invalidates the transaction.

The decision to cancel the changes shall be published when the context changes are to be discarded.

If the decision is to accept the changes, the value of the output `decision` parameter is “accept”. If the decision is to cancel the changes, the value of the output `decision` is “cancel”. The context manager is shall treat these values in a case-insensitive manner.

Once the decision has been published, the change transaction is complete.

The exception InvalidContextCoupon is raised if the input `contextCoupon` does not denote the transaction currently in progress.

The exception NotInTransaction is raised if there is no change transaction currently in progress.

The exception ChangesNotEnded is raised if the method EndContextChanges has not yet been performed during the course of the current transaction.

The exception AcceptNotPossible is raised if the decision to be published is `accept` but there were applications for which it was not possible to obtain a survey response (e.g., these applications were blocked). The decision `accept` in this case is erroneous. This exception defends against this case should it arise due to an application programming error.
15.3.5.9 SuspendParticipation

This method enables an application to indicate that it wants to suspend its active participation in a common context system while remaining registered as a participant. An application that is suspended will not be informed about context changes, and does not need to remain in synchrony with the context. The specific circumstances during which an application may suspend its participation are described below.

Use of this method also enables an application to ensure that the context manager does not give up the application’s slot in the common context to another application. Context managers can be implemented to support a maximum number of participants. If an application leaves a context system, it risks not being able to rejoin. In contrast, by suspending its participation, this possibility is avoided.

An application may suspend its participation as an alternative to leaving the common context (see Section 15.3.5.4 LeaveCommonContext). When an application suspends its participation for this reason this manner, as the direct result of an explicit user command, it shall behave exactly as though it has broken its link with the context system. For example, the application shall clearly indicate to the user that its link is broken. An application that is suspended shall still explicitly leave the context system when the application terminates.

In the absence of an explicit user command, an application may still suspend its participation. An application may elect to suspend its participation in order to minimize its use of computational resources. This might occur when the application is in a state such that responding to a context change does not provide any benefit to the user. For example, an application might suspend its participation when its display is minimized and therefore cannot be seen by the user.

An application that is decides to suspend itself (i.e., without an explicit command from the user) is still considered to be linked even though it is not tracking context changes. This is because the only way an application’s link can be broken is when the user explicitly indicates to the application that this be done. A self-suspended application shall continue to indicate that it is linked, but it shall not display data that is context-sensitive. For example, an application might ensure that its data display is not visible (i.e., the display is minimized) while it is suspended.

An application that wants to suspend itself denotes itself with its participant coupon as the value of the input ParticipantCoupon.

A suspended application can subsequently resume its participation in the common context via the ContextManager::ResumeParticipation method. The application will not be surveyed, nor will it be informed of changes to the common context until the it invokes the ContextManager::ResumeParticipation method.
In order to avoid a deadlock condition, this method does not block. If this method was allowed to block, it would be possible for an application to block while the context manager was attempting to perform a method on the application’s ContextParticipant interface. For single-threaded applications, this could cause a deadlock.

Consequently, if a context change transaction is in progress when this method is called, the application may still be notified about the context change. The application is free to ignore this notification or may not even be capable of responding. The context manager will robustly handle the failure of an application to respond.

This method has no effect if the application has already suspended its participation.

A suspended application cannot instigate a context change transaction.

A suspended application will be informed about the termination of the common context system should this occur while the application is suspended (see Section 15.3.6, ContextParticipant (CP)).

Context manager implementations are encouraged to periodically confirm that suspended context participants are still running. This is to avoid the situation in which context manager continues to allocate internal resources to a suspended participant that subsequently fails without first informing the context manager that it is leaving the common context system.

The exception UnknownParticipant is raised if the input participantCoupon does not denote an application that is currently a participant in the common context system.

15.3.5.10 ResumeParticipation

This method enables a suspended application to indicate that it wants to resume active participation in a common context system. The application denotes itself with its participant coupon as the value of the input participantCoupon. Upon resuming, an application must automatically ensure that it has established synchrony with the current context. It can either set its internal state to match the current context, or it can set the current context to match its internal state.

The application denotes itself with its participant coupon. This method has no effect if the application did not previously invoke the method ContextManager::SuspendParticipation.

An application can only resume its participation a common context system between context change transactions. If no transaction is in progress, the application is able to immediately resume participation in the context change system.

If a transaction is in progress and the value of the input wait indicates true, this method will block until the transaction completes. It is recommended that an application that is willing to wait also display a message to the user indicating that it is attempting to resume participation.
in a common context system. If a transaction is in progress and the value of the input `wait`
indicates false, this method immediately raises the exception `TransactionInProgress`.

The exception `UnknownParticipant` is raised if the input `participantCoupon` does not denote
an application that is currently a participant in the common context system.
15.3.6 ContextParticipant (CP)

```java
interface ContextParticipant {
    ContextChangesPending
    inputs(long contextCoupon)
    outputs(string decision, string reason)
    raises()

    ContextChangesAccepted
    inputs(long contextCoupon)
    outputs()
    raises()

    ContextChangesCanceled
    inputs(long contextCoupon)
    outputs()
    raises()

    CommonContextTerminated
    inputs()
    outputs()
    raises()

    Ping
    inputs()
    outputs()
    raises()
}
```

15.3.6.1 Synopsis

This interface enables a context participant to be notified about context changes that it did not initiate.

15.3.6.2 ContextChangesPending

This method informs a participant in a common context system that a change to the common context data is pending. The value of the input `contextCoupon` denotes the transaction within which the context changes occurred. The participant shall respond with an indication of how it wants to deal with the change:

- Accept the change
- Conditionally accept the change (e.g., because it is in the middle of a task that would cause significant user work to be lost if a context change was allowed)

An application that accepts the changes is willing to apply the new context data if subsequently instructed to do so (by the `ContextParticipant::ContextChangesAccepted` or `ContextParticipant::ContextChangesCanceled` methods). For an application that accepts the
changes, the value of the output \textit{decision} shall be “accept” and the value of the output \textit{reason} shall be an empty string (“”).

An application that conditionally accepts the changes is also willing to apply the changes, but only after informing the user that the application might lose work that the user is in the midst of performing. For an application that accepts the changes, the value of the output \textit{decision} shall be “conditionally_accept”. The output \textit{reason} shall contain a succinct but informative description of the work that might be lost. (The description shall not identify the application as this information is provided by the application when it joins the common context system.) The application through which the user instigated the context changes is responsible for informing the user of the situation and obtaining the user’s decision about how to proceed.

An application that cannot interpret the context data (e.g., does not know who the patient is) shall accept the changes. However, the application shall clearly indicate to the user (e.g., by displaying a message) that it cannot apply the current context data.

The context manager shall treat the value of the output \textit{decision} in a case-insensitive manner. The context manager shall preserve the case used in the value of the output \textit{reason}.

If a participant does not respond in a timely manner, it will be interpreted by the context manager as being busy. The amount of time that the manager will wait before determining that an application is busy depends upon the manager’s implementation. This method is not performed upon the application that instigated the context changes. Instead, the application is blocked by the manager when it performs ContextManager::EndContextChanges.

\textbf{15.3.6.3 ContextChangesAccepted}

This method informs a participant in a common context system that the result of the most recent context change survey was to accept the changes and that the common context data has indeed been set. The participant can access the context data via the context manager’s ContextData interface to obtain the changes. The value of the input \textit{contextCoupon} denotes the transaction within which the context changes occurred. This coupon is needed in order to access the context data.

If it is not possible to perform this method on an application because it is busy, the context manager will periodically keep trying until it has successfully performed the method, or until a new context change transaction is initiated. The intervals at which the context manager tries to retry this method is implementation-dependant.
15.3.6.4 ContextChangesCanceled

This method informs a participant in a common context system that a context change transaction has been canceled. The value of the input contextCoupon denotes the transaction that has been canceled.

If it is not possible to perform this method on an application because it is busy, the context manager will periodically keep trying until it has successfully performed the method, or until a new context change transaction is initiated. The intervals at which the context manager tries to retry this method is implementation-dependant.

15.3.6.5 CommonContextTerminated

This method informs a participant in a common context system that the system is being terminated. The participant will not be subsequently informed about context changes, nor will it be able to perform common context changes. The participant shall dispose of any context manager interface references it holds. The participant shall not perform any other methods upon the context manager prior to performing the ContextManager::JoinCommonContext to establish its participation in a new instance of a common context system.

15.3.6.6 Ping

This method provides a means for a context manager to determine whether or not a participant in a common context system is still running. This method shall be implemented by all participants to return immediately. The context manager can then perform this method to probe a participant when its existence is in question.

In performing this method, the context manager will be able to indirectly exercise the underlying communications infrastructure. The infrastructure will either indicate that the method was successfully performed, that the method failed because the participant no longer exists, or that the method failed but it cannot be determined whether or not the participant exists. In this last case, the manager shall assume that the participant still exists.
15.3.7 ImplementationInformation (II)

interface ImplementationInformation {
    readonly string ComponentName
    readonly string RevMajorNum
    readonly string RevMinorNum
    readonly string PartNumber
    readonly string Manufacturer
    readonly string TargetOS
    readonly string TargetOSRev
    readonly string WhenInstalled
}

15.3.7.1 Synopsis
This interface enables a component to expose information pertaining to its implementation.

15.3.7.2 ComponentName
This read-only property is the name of the component, specifically, “Patient Link Mapping Agent”.

15.3.7.3 RevMajorNum
This read-only property is the major number for the software revision for the component, as assigned by its manufacturer. For example, in the full revision number Z.32, ‘Z’ is the major number and might indicate a particular functional release of the software.

15.3.7.4 RevMinorNum
This read-only property is the minor number of the software revision for the component, as assigned by its manufacturer. For example, in the full revision number Z.32, ‘32’ is the minor number and might indicate a particular build of the software.

15.3.7.5 PartNumber
This read-only property is the part number that the component’s manufacturer assigned to the component.

15.3.7.6 Manufacturer
This read-only property is the name of the organization that developed the component.

15.3.7.7 TargetOS
This read-only property is the name of the operating system on which the component is able to execute.
15.3.7.8 TargetOsRev
This read-only property is the revision of the operating system named in target operating system on which the component is able to execute.

15.3.7.9 WhenInstalled
This read-only property is the date and time at which the component was installed on its host.
### 15.3.8 MappingAgent (MA)

```java
interface MappingAgent {
    ContextChangesPending
    inputs(long mappingAgentCoupon, Principal contextMgr,
           long contextCoupon)
    outputs(string decision, string reason)
    raises()

    Ping
    inputs()
    outputs()
    raises()
}
```

#### 15.3.8.1 Synopsis

This interface enables a mapping agent to be notified about context changes. A mapping agent may implement the interface ContextAgent (See Section 15.3.3) as an alternative to this interface.

#### 15.3.8.2 ContextChangesPending

This method informs a mapping agent in a common context system that a change to the common context data is pending. The value of the input `contextCoupon` denotes the transaction within which the context changes occurred. The value of the input `mappingAgentCoupon` is a predefined coupon that denotes the specific type of mapping agent. (See Section 15.2.10 Pre-Defined Context Agent Coupons). The value of the input `contextMgr` is an interface reference to the context manager’s principal interface. This is so that the mapping agent can easily obtain the context manager interface(s) it needs.

The agent shall respond with an indication of how it wants to deal with the context change:

- The changes are valid
- The changes are invalid

If the changes are valid, then the value of the output `decision` shall be “valid”. If the changes are invalid, then the value of the output `decision` shall be “invalid”. The changes shall only be declared invalid if the set of identifiers in the proposed context data do not all represent the same real-world entity or concept. If the changes are invalid, then the value of the output `reason` shall contain a succinct but detailed string describing why the changes were invalid. Otherwise the value of `reason` shall be an empty string (""").

The context manager shall treat the value of the output `decision` in a case-insensitive manner. The context manager shall preserve the case used in the value of the output `reason`. 


15.3.8.3 Ping

This method provides a means for a context manager to determine whether or not a mapping agent in a common context system is still running. This method shall be implemented by all agents to return immediately. The context manager can then perform this method to probe a mapping agent when the agent’s existence is in doubt.

In performing this method, the context manager will be able to indirectly exercise the underlying communications infrastructure. The infrastructure will either indicate that the method was successfully performed, that the method failed because the agent no longer exists, or that the method failed but it cannot be determined whether or not the agent exists. In this last case, the manager shall assume that the agent still exists.
15.3.9 SecureBinding (SB)

```java
interface SecureBinding {
    exception UnknownBindee {}
    exception UnknownPropertyName { string propertyName; }
    exception BadPropertyType { string propertyName; type actual; type expected; }
    exception BadPropertyValue { string propertyName; variant itemValue; string reason; }
    exception NameValueCountMismatch { long numNames; long numValues }
    exception ImproperKeyFormat { string reason; }
    exception ImproperMACFormat { string reason; }
    exception BindingRejected { string reason; }
    exception AuthenticationFailed { string reason; }

    InitializeBinding
    inputs(long bindeeCoupon, string[] propertyNames, variant[] propertyValues)
    outputs(string binderPublicKey, string mac)
    raises(UnknownBindee, NameValueCountMismatch, UnknownPropertyName, BadPropertyType, BadPropertyValue, BindingRejected)

    FinalizeBinding
    inputs(long bindeeCoupon, string bindeePublicKey, string mac)
    outputs(string[] privileges)
    raises(UnknownBindee, ImproperKeyFormat, ImproperMACFormat, AuthenticationFailed)
}
```

15.3.9.1 Synopsis

This interface enables a component to exchange security-related credentials with another component for subsequent use in interactions that need to be secure.

15.3.9.2 InitializeBinding

This method enables a context management component ("bindee") to initiate the process of establishing a secure binding with another context management component ("binder"). The bindee shall complete the process of establishing a secure binding with the binder by performing the method SecureBinding::FinalizeBinding upon the binder.

A secure binding shall be established by the bindee before it attempts to interact with the binder via methods that entail the use of either the bindee’s or the binder’s digital signature. For example, an application or user mapping agent shall establish a secure binding with the context manager before it attempts to access the context manager in order to set or get context item values that require the bindee’s digital signature. An application shall establish
a secure binding with the authentication repository before attempting to set or get user
authentication data from the authentication repository.

This method shall be performed only after the bindee has been provided by the binder with a
coupon to denote itself. The value of the input \textit{bindeeCoupon} is this coupon. The value of
\textit{bindeeCoupon} depends upon the role bindee and binder, as described below:


\begin{tabular}{|l|l|l|}
\hline
Bindee & Binder & Value of \textit{bindeeCoupon} \\
\hline
Context & Context & Participant coupon, obtained by the participant from the context manager via ContextManager::JoinCommonContext. \\
Participant & Manager & \\
Application & & \\
Context & Authentication & Connection coupon, obtained by the participant from the authentication repository via AuthenticationRepository::Connect. \\
Participant & Repository & \\
Application & & \\
Context & Context & Context agent coupon, based upon the \textit{type of mapping agent}, as defined in Section \ref{sec:contextagentcoupons} \\
Agent & Manager & \textbf{Pre-Defined Context Agent Coupons} \\
\hline
\end{tabular}

As part of the process of establishing a secure binding, it is necessary for the bindee and the
binder to agree upon the properties of the underlying security algorithms that they will use in
subsequent secure interactions. These properties may include the public key / private key
scheme, the number of bits used to represent a key, and the type of one-way hash algorithm
that is to be used to generate message digests and message authentication codes. The specific
properties that must be agreed upon, and the allowed set of values for these properties, are
defined in the each of the HL7 context management technology-specific component mapping
specification documents.

The value of the input sequence \textit{propertyNames} contains the names of the secure binding-
related properties for which the bindee wishes to establish agreement. The values for each of
these properties are contained in the input sequence \textit{propertyValues}. The \textit{i}th element in
\textit{propertyValues} is the value for the property named by the \textit{i}th element in \textit{propertyNames}. The
data type for a property is the same as the data type of the element in \textit{propertyValues} that
contains the property’s value.

The value of the output \textit{binderPublicKey} is the binder’s public key, and shall be used by the
bindee in all subsequent secure interactions that involve the binder. The value of
\textit{binderPublicKey} is character-encoded binary data formed by the binder when it computes its
public key / private key pair.
The value of the output \textit{mac} is the message authentication code. This code shall be used by the bindee to prove the identity of the binder, and to ensure that the value of \textit{binderPublicKey} has not been tampered with.

The value of \textit{mac} is character-encoded binary data formed by the binder’s computation of a one-way hash value. This hash value is computed using an input string formed by concatenating the bindee’s passcode to the end of the character-encoded binary string containing the binder’s public key. This passcode is a secret known only to the bindee and the binder. Upon receipt of the output \textit{mac} and \textit{binderPublicKey}, the bindee independently creates the same string as the binder and performs the same hash computation. If the resulting hash value matches the value of \textit{mac}, then the binder shall be considered authentic and the value of \textit{binderPublicKey} shall be considered valid.

The algorithms used to compute \textit{mac} and \textit{binderPublicKey} are technology-specific. The format of these outputs are also technology specific.

The exception \textit{UnknownBindee} is raised if the input \textit{bindeeCoupon} does not denote a context management component currently known to the binder.

The exception \textit{NameValueCountMismatch} is raised if the number of items in the input \textit{propertyNames} does not match the number of items in the input \textit{propertyValues}.

The exception \textit{BadPropertyType} is raised if the data type for one or more of the properties whose value is to be set is not the same as the expected data type.

The exception \textit{BadPropertyValue} is raised if the data value for one or more of the properties whose value is to be set is determined to be unacceptable or incompatible.

The exception \textit{BindingRejected} is raised if the binder is unable to establish a secure binding with the bindee. For example, the binder raises this exception if it does not know the bindee’s passcode.

\textbf{15.3.9.3 FinalizeBinding}

This method enables bindee to finalize the process of establishing a secure binding with a context management component, and enables the bindee to determine what access privileges it has. This method shall be performed by a bindee only after it has successfully performed the method \textit{InitializeBinding} upon a binder. The bindee denotes itself using the same value for the input \textit{bindeeCoupon} that it used when it performed the method \textit{InitializeBinding} upon the binder.

The input \textit{bindeePublicKey} is the bindee’s public key, and shall be used by the binder in all subsequent secure interactions that involve the bindee. The value of \textit{bindeePublicKey} is character-encoded binary data formed by the bindee when it computes its public key / private key pair.
The input \textit{mac} is the message authentication code. This code shall be used by the binder to prove the identity of the bindee, and to ensure that the value of \textit{bindeePublicKey} has not been tampered with.

The value of \textit{mac} is character-encoded binary data formed by the bindee’s computation of a one-way hash value. This hash value is computed using an input string formed by concatenating the bindee’s passcode to the end of the character-encoded binary string containing the bindee’s public key. This passcode is a secret known only to the bindee and the binder. Upon receipt of the inputs \textit{mac} and \textit{bindeePublicKey}, the binder independently creates the same string as the bindee and performs the same hash computation. If the resulting hash value matches the value of \textit{mac}, then the bindee shall be considered authentic and the value of \textit{bindeePublicKey} shall be considered valid.

The algorithms used to compute \textit{mac} and \textit{bindeePublicKey} are technology-specific. The format of these inputs are also technology specific.

The sequence output \textit{privileges} describes the bindee’s access privilege in a manner that depends upon the type of component with which the binding has been established.

When implemented by the context manager, this sequence indicates on a per-subject basis whether or not the bindee is allowed to get and/or set items within a subject, for example, via the context manager’s SecureContextData interface. For each subject for which the bindee has access privileges, the name of the subject appears as the \textit{i}th element in the sequence, and a string indicating the bindee’s access privilege for this subject appears as the \textit{i+1}st element.

The following types of access privileges are defined for the context manager:  

- “Any” means that subject is a common subject, and access privileges are not enforced. The bindee may get or set the items in the subject, and the bindee’s digital signature is not required to do so.
- “None” means that the bindee can neither set nor get the items in the subject.

\footnote{In CMA Version 1.2, only two privileges were defined for the context manager: “Get” and “Set”. However, with the introduction of authenticated gets (See 15.3.10.4, GetItemValues), it is necessary to introduce the “Geta” and “Seta” privileges. This allows the privileges “Get” and “Set” to maintain the meaning that was specified in V1.2. Further, if a bindee did not have any access privileges for a subject, then, in V1.2, that subject did not appear in the list of subjects included in the output \textit{privileges}. Now, that subject will appear in the list but with an access privilege of “None”. In V1.2, in the absence of authenticated gets, it was not possible to actually define a subject for which a bindee did not have any access privileges, so the change in how this is represented will not affect existing applications. Finally, in V1.2, it was not possible for an application to determine if a subject was a secure subject or a common subject. Now a common subject will appear in the list of subjects with an access privilege of “Any”. By definition, a subject with any other type of access privilege is a secure subject.}
“Get” means that the bindee can get, but not set, items in the subject. The bindee’s digital signature is not required to get the items in the subject.

“Geta” means that the bindee can get, but not set, items in the subject. The bindee’s digital signature is required to get the items in the subject, so that the bindee may be authenticated by the context manager.

“Set” means that the bindee can set as well as get items in the subject. The bindee’s digital signature is not required to get or set the items in the subject.

“Seta” means that the bindee can set as well as get items in the subject. The bindee’s digital signature is required to get or set the items in the subject, so that the bindee may be authenticated by the context manager.

An example of the elements in the sequence privileges is:

```
“Patient”
“Any”
“User”
“Get”
```

The strings representing the subject name and access privileges are case insensitive.

When implemented by the authentication repository, the output sequence privileges indicates whether or not the bindee is allowed to get and/or set user authentication data for the application whose user data is maintained by the repository. A sequence with at most one element is returned. This element is a string that indicates whether or not the bindee is allowed to get and/or set user authentication data for the application. The following types of access privileges are defined for the authentication repository:

- “None” means that the bindee can neither set nor get the user authentication data.
- “Get” means that the bindee can get, but not set, the user authentication data. The bindee’s digital signature is required to get the user authentication data, so that the bindee may be authenticated by the authentication repository.

---

13 These privileges are the same as specified in the CMA Version 1.2. The descriptions for these privileges have been elaborated but the meaning is also unchanged. It is understood that the privilege “Get” for the authentication repository has the same meaning as “Geta” for the context manager, and that “Set” for the authentication repository has the same meaning as “Seta” for the context manager. The names for the authentication repository’s access privileges are unchanged relative to the new privileges introduced for the context manager in order to preserve backward compatibility.
“Set” means that the bindee can set as well as get the user authentication data. The bindee’s digital signature is required to get or set the user authentication data, so that the bindee may be authenticated by the authentication repository.

The strings representing access privileges are case insensitive.

The exception UnknownBinding is raised if the input bindingCoupon does not denote an bindee currently known to the binder.

The exception ImproperKeyFormat is raised if the input publicKey is not properly formatted.

The exception ImproperMACFormat is raised if the input mac is not properly formatted.

The exception AuthenticationFailed is raised if the input mac does not establish the identity of the bindee and/or the integrity of the input bindeePublicKey.
15.3.10 SecureContextData (SD)

interface SecureContextData {
    exception NotInTransaction()
    exception InvalidContextCoupon()
    exception UnknownParticipant(long participantCoupon)
    exception UnknownItemName(string itemName)
    exception BadItemNameFormat(string itemName, string reason)
    exception BadItemType(string itemName, string actual, string expected)
    exception BadItemValue(string itemName, variant itemValue)
    exception NameValueCountMismatch(long numNames, long numValues)
    exception ChangesNotPossible()
    exception ChangesNotAllowed()
    exception SignatureRequired()
    exception AuthenticationFailed(string reason)

    GetItemNames
        inputs(long contextCoupon)
        outputs(string[] itemNames)
        raises(InvalidContextCoupon)

    SetItemValues
        inputs(long participantCoupon, string[] itemNames,
                variant[] itemValues, long contextCoupon, string appSignature)
        outputs()
        raises(NotInTransaction, InvalidContextCoupon, UnknownParticipant,
                NameValueCountMismatch, BadItemNameFormat, BadItemType,
                BadItemValue, ChangesNotPossible, ChangesNotAllowed,
                SignatureRequired, AuthenticationFailed)

    GetItemValues
        inputs(long participantCoupon, string[] itemNames,
                boolean onlyChanges, long contextCoupon, string appSignature)
        outputs(variant[] itemValues, string managerSignature)
        raises(InvalidContextCoupon, UnknownParticipant,
                BadItemNameFormat, UnknownItemName, SignatureRequired,
                AuthenticationFailed)
    }

15.3.10.1 Synopsis

This interface enables a context participant to securely get and set context data. The data is represented as a set of items, each of which is structured as a name/value pair. The context data for all subjects – common as well as secure - can be accessed via this interface.

15.3.10.2 GetItemNames

This method is identical to ContextData::GetItemNames.
15.3.10.3 SetItemValues

This method is similar to ContextData::SetItemValues. The primary difference is that the context participant’s digital signature shall be provided as the value of the input appSignature when secure subject item values are among the items to be set. This signature enables the context manager to authenticate that they came from a designated application or from a valid secure subject mapping agent, and that the values were not tampered with between the time they were sent and were received.

A signature is not required when the values for the user subject items are null. This enables any application to set the user context to empty. When a signature is not provided, the value of the input appSignature shall be an empty string (""").

Concatenating the string representations of the following inputs in the order listed shall form the data from which a message digest is computed by the participant:

- participantCoupon
- itemNames (i.e., All the elements in the order that they appear in the array.)
- itemValues (i.e., All the elements in the order that they appear in the array.)
- contextCoupon

A participant shall compute its digital signature by encrypting the message digest with its private key.

The exception SignatureRequired is raised if the value of appSignature is not a digital signature and a signature is required in order to perform this method.

The exception AuthenticationFailed is raised if a digital signature is required and provided, but the process of authentication determines that: the application that invoked this method did not previously provide its public key via the interface SecureBinding; that the input appSignature has been forged; that the input parameter values have been tampered with; that the participant has not been designated for performing user context changes.

15.3.10.4 GetItemValues

This method is similar to ContextData::GetItemValues. The primary difference is that context participant may need to provide its digital signature as an input in order to access the items named for retrieval, and the context manager may need to provide its digital signature as an output along with the items retrieved.

The inclusion of the context participant’s signature as an input enables the context manager to authenticate the context participant for secure subjects whose access privileges require doing so. (See Section 15.3.9.3 FinalizeBinding) The inclusion of the context manager’s
A context participant shall provide its digital signature as the value of the input \texttt{appSignature} whenever it lists for retrieval the name of at least one item belonging to a subject for which the context participant’s access privilege is either “Geta” or “Seta”. (See Section \ref{finalization}) Under these circumstances, the context participant shall also provide its participant coupon as the value of the input \texttt{participantCoupon}. Otherwise, value of the input \texttt{appSignature} input shall be an empty string (“”) and the value of the input \texttt{participantCoupon} shall be zero (0).

Concatenating the string representations of the following inputs in the order listed shall form the data from which a message digest is computed by the context participant:

- \texttt{participantCoupon}
- \texttt{itemNames} (i.e., All the elements in the order that they appear in the array.)
- \texttt{onlyChanges}
- \texttt{contextCoupon}

The context participant shall compute its digital signature by encrypting the message digest with its private key.

The context manager shall provide its digital signature as the value of the output \texttt{managerSignature} whenever at least one of the items named for retrieval belongs to a secure subject. Concatenating the string representations of the following output and input in the order listed shall form the data from which a message digest is computed by the context manager:

- \texttt{itemValues} (i.e., All the elements in the order that they appear in the array.)
- \texttt{contextCoupon}

The context manager shall compute its digital signature by encrypting the message digest with its private key.

The exception \texttt{SignatureRequired} is raised if the value of \texttt{appSignature} is not a digital signature and a signature is required to perform this method.

The exception \texttt{AuthenticationFailed} is raised if a digital signature is required and provided, but the process of authentication determines that: the application that invoked this method did not previously provide its public key via the interface SecureBinding; that the input
appSignature has been forged; that the input parameter values have been tampered with; that
the participant is not allowed to access the requested context items.
16  Conformance

The HL7 Context Management Standard, Version 1.3, (“the Standard”) is embodied in the following documents:

- Health Level-Seven Standard Context Management Specification, Technology and Subject-Independent Component Architecture, Version CM-1.3 (i.e., this document), called “the Architecture Specification” in this section.

- Health Level-Seven Standard Context Management Specification, Subject Data Definitions, Version CM-1.3, called “the Subject Specification” in this section.


The responsibilities for applications and CMA components relative to claims of conformance with the Standard are described below.

16.1  General Approach and Goals

Application standards, particularly those in healthcare, often represent a trade-off between “plug and play” interoperability and adaptability to a wide variety of user requirements. The Context Management Standard was conceived and is specified in order to be very nearly plug and play.

Specifically, the Standard contains very few options. Where there are options, vendors are generally expected to implement all the options, so that users are not surprised by compatibility issues after contracting for a product. In the few cases where there are optional features that need not be adopted by a vendor in order to claim compliance, a concise conformance form, defined later in this section, shall be used by vendors to unambiguously describe their conformance with the Standard so that purchasers may understand this a priori.
The following sections apply to the various kinds of programs that have categorically different requirements for conformance:

- Application
- Context Manager
- Mapping Agent
- Annotation Agent
- Authentication Repository

The sections below describe the requirements for conformance for each of these types of programs.

16.1.1 Context Management Standard Declaration of Conformance

For each program category listed above a Context Management Standard Declaration of Conformance Form is defined. Vendors of products in one of the program categories should use to certify the conformance of those products. Purchasers should be able to receive a copy of the form, without modifications to the wording, signed by a representative of the vendor with the authority to enter into contracts, prior to contracting for the purchase of software for which conformance to the Standard is claimed.

16.1.2 Future Conformance Requirements

It is not expected that products in all program categories will always be required to conform to all of the standard subjects or technology mappings defined in future versions of the Standard. As future versions of the Context Management Standard define new subject areas and technology mappings, the revised standards will include revised Declaration Forms, so that vendors can indicate the new subjects and/or technologies that they do support.

The table below indicates the expectations with respect to current and future conformance requirements.

<table>
<thead>
<tr>
<th>Product Category</th>
<th>Technology Mappings</th>
<th>Standard Subject Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>declare those supported</td>
<td>declare those supported</td>
</tr>
<tr>
<td>Context Manager</td>
<td>declare those supported</td>
<td>support all</td>
</tr>
<tr>
<td>Mapping Agent</td>
<td>declare those supported</td>
<td>declare those supported</td>
</tr>
<tr>
<td>Annotation Agent</td>
<td>declare those supported</td>
<td>declare those supported</td>
</tr>
<tr>
<td>Authentication Repository</td>
<td>declare those supported</td>
<td>n/a</td>
</tr>
</tbody>
</table>
16.2 Applications

Any application claiming to conform to the Standard shall provide the following certificate and conform to the requirements that are checked.

---

Health Level Seven Context Management Standard

Declaration of Conformance

Version 1.3

Application

It is required that all the boxes below be checked.

- Implements all of the policies, protocols, rules, and constraints defined for applications as specified in the Architecture Specification; Implements the ContextParticipant interface, and uses the ContextManager interface as well as the ContextData, SecureContextData, and SecureBinding interfaces as necessary as defined in the Architecture Specification.

- Implements the user interface policies, protocols, rules, and constraints defined for applications and as specified in the User Interface Specification.

At least one of the boxes below must be checked.

- Implements the Patient Link capability as defined for applications in the Architecture Specification and Subject Specification.

- Implements the Encounter Link capability as defined for applications in the Architecture Specification and Subject Specification.

- Implements the Observation Link capability as defined for applications in the Architecture Specification and Subject Specification.

- Implements the User Link capability as defined for applications in the Architecture Specification and Subject Specification.

- Implements the Certificate Link capability as defined for applications in the Architecture Specification and Subject Specification.

The following boxes below may be checked.

- Implements the Custom Link capability for one or more custom subjects, as defined for applications in the Architecture Specification and Subject Specification.

- Uses an authentication repository, including the AuthenticationRepository interface, as defined in the Architecture Specification.

At least one of the boxes below must be checked.

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for applications and specified in the ActiveX Technology Mapping.

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for applications and specified in the Web Technology Mapping.
## 16.3 Context Manager

Any context manager claiming to conform to the Standard shall provide the following form, and conform to the statements that it contains.

### Health Level Seven Context Management Standard

#### Declaration of Conformance

#### Version 1.3

#### Context Manager

**It is required that all the boxes below be checked.**

- Implements all of the policies, protocols, rules, and constraints defined for context managers as specified in the Architecture Specification; Implements the ContextManager, ContextData, SecureContextData, SecureBinding, and ImplementationInformation interfaces; uses the ContextParticipant, MappingAgent and ContextAgent interfaces, as defined in the Architecture Specification.

- Implements general support for common and secure links, as well as specific support for Patient Link, Encounter Link, Observation Link, User Link, Certificate Link, and multiple custom links, as specified in the Architecture Specification and Subject Specifications.

- For each of the technologies supported (see below), implements full context sharing and interoperability amongst and between the applications and mapping agents that employ these technologies.

**At least one of the boxes below must be checked.**

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for context managers and specified in the ActiveX Technology Mapping.

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for context managers and specified in the Web Technology Mapping.
16.4 Mapping Agent

Any mapping agent claiming to conform to the Standard shall provide the following certificate and conform to the requirements that are checked.

Health Level Seven Context Management Standard

Declaration of Conformance

Version 1.3

Mapping Agent

It is required that all the boxes below be checked.

- Implements all of the policies, protocols, rules, and constraints defined for a mapping agent as specified in the Architecture Specification.

- Implements the MappingAgent or ContextAgent interface, and the ImplementationInformation interface, and uses the SecureBinding, ContextData or SecureContextData interfaces as necessary as specified in the Architecture Specification.

At least one of the boxes below must be checked.

- Implements support for a Patient Mapping Agent.

- Implements support for an Encounter Agent.

- Implements support for an Observation Agent.

- Implements support for a User Mapping Agent.

- Implements support for a Custom Mapping Agent.

At least one of the boxes below must be checked.

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for mapping agents and specified in the ActiveX Technology Mapping.

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for mapping agents and specified in the Web Technology Mapping.
16.5 **Annotation Agent**

Any annotation agent claiming to conform to the Standard shall provide the following certificate and conform to the requirements that are checked.

---

**Health Level Seven Context Management Standard**

**Declaration of Conformance**

**Version 1.3**

**Annotation Agent**

It is required that all the boxes below be checked.

- **☐** Implements all of the policies, protocols, rules, and constraints defined for an annotation agent as specified in the Architecture Specification.

- **☐** Implements the ContextAgent and ImplementationInformation interfaces and uses the SecureBinding, ContextData or SecureContextData interfaces as necessary as specified in the Architecture Specification.

At least one of the boxes below must be checked.

- **☐** Implements support for a Certificate Annotation Agent.

- **☐** Implements support for a Custom Annotation Agent.

At least one of the boxes below must be checked.

- **☐** Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for annotation agents and specified in the ActiveX Technology Mapping.

- **☐** Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for annotation agents and specified in the Web Technology Mapping.
16.6 **Authentication Repository**

Any authentication repository claiming to conform to the Standard shall provide the following certificate and conform to the requirements that are checked.

---

**Health Level Seven Context Management Standard**

**Declaration of Conformance**

**Version 1.3**

**Authentication Repository**

*It is required that all the boxes below be checked.*

- Implements all of the policies, protocols, rules, and constraints defined for an authentication repository as specified in the Architecture Specification; implements the AuthenticationRepository, SecureBinding, and ImplementationInformation interfaces as specified in the Architecture Specification.

*At least one of the boxes below must be checked.*

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for authentication repositories and specified in the ActiveX Technology Mapping.

- Employs the technology, subject to all of the policies, protocols, rules, and constraints, defined for authentication repositories and specified in the Web Technology Mapping.
17 Backwards Compatibility

The HL7 Context Management Architecture specified in this document, which is Version 1.3, is fully backwards compatible with Version 1.2 of the Architecture.
Appendix: Diagramming Conventions

There are four types of formal diagrams that are used throughout this document to describe the CMA architecture:

- A use case diagram depicts the actors (human and/or computer-based) and the roles that they play when participating in an interesting scenario.
- A use case interaction diagram illustrates the high-level interactions between the actors that participate in the use case.
- A component architecture diagram depicts components and their interfaces, and indicates the interfaces each component uses for communicating with other components.
- A component interaction diagram illustrates the series of method invocations that components perform on each other in order to implement a particular use case.

The conventions for each of these diagrams are explained below. The motivation for these types of diagrams may be found in Ivar Jacobson’s text *Object-Oriented Software Engineering*. The specific diagramming conventions are consistent with those defined for the Unified Modeling Language*.

Use Case Diagram

The use case diagramming conventions are:

- A stick figure represents an actor, even if the actor is a computer-based entity, such as an application:

![Healthcare Application](image-url)

---


• An oval represents a use case. The name of the use case appears next to the oval:

![Patient Selection Change](image)

• An arrow directed from an actor to the use case indicates that the actor participates in the use case. A label near the arrow succinctly describes the actor’s role in the use case:

![Participates](image)

**Use Case Interaction Diagrams**

The use case interaction diagramming conventions are:

• The interacting actors are depicted by rectangles labeled with the actor’s name, arranged in a horizontal row. A vertical dashed bar descends from each of these rectangles:

![User](image)

• An interaction that is initiated by an actor is represented as an arrow that emanates from the actor. The arrow terminates on the actor to which the interaction is directed. Each arrow is labeled with a short description of the interaction it represents:

![User](image) ![Application XXX](image)

I choose patient “Sam Smith”
• A vertical bar indicates the start and end of the actions that an actor performs in response to an interaction. These actions may include additional interactions:

```
User

I choose patient “Sam Smith”
```

• An actor can respond to an interaction. A response is shown as an arrow labeled with an indication of the response:

```
Context Manager

I choose patient “Sam Smith”
```

```
Application XXX

The selected patient is now “Sam Smith”
```

• The entire set of interaction arrows is temporally ordered, from left to right, top to bottom.
Component Architecture Diagrams

The component architecture diagramming conventions are:

- Each component is depicted as a rectangle. The name of the component appears within the rectangle:

  ![Context Manager Diagram](image)

- Each of the interfaces implemented by a component is illustrated as a circle tangent to the rectangle that depicts the component. Each circle is labeled with the name of the interface it represents. Two or three letter abbreviations are typically used:

  ![Interface Diagram](image)

- A directed arrow connects components that communicate with each other. Arrows emanate from a client component and point to the server components that it uses. Each arrow terminates on the circle representing the specific server component interface that is used. A distinct arrow is used for each interface for each server component that a client component uses:

  ![Communication Diagram](image)
Component Interaction Diagrams

The component interaction diagramming conventions are:

- The interacting components are depicted by rectangles labeled with the component’s name, arranged in a horizontal row. A vertical dashed bar descends from each of these rectangles:

```
  Context Manager
```

- A method that is invoked by a component is represented as an arrow that emanates from the bar and that terminates on the bar for the component that services the method. Each arrow is labeled with the name of the method it represents. Examples of actual parameter values may be included for clarity:

```
  Application XXX
  Context Manager

  PublishChangesDecision("accept")
```

- A vertical bar indicates the start and end of the processing that a component performs in response to a method invocation. This processing may itself include method invocations:

```
  Application XXX
  Context Manager

  PublishChangesDecision("accept")
```
• Method return values are indicated when this aids in understanding the use case. A return value is shown as an arrow labeled with an indication of the return value:

• The entire set of method invocation arrows is temporally ordered, from left to right, top to bottom.
# Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>An application’s response when it is willing to accept the context data changes and to change its internal state accordingly if the changes are published.</td>
</tr>
<tr>
<td>Accept-Conditional</td>
<td>An application’s response when it is in the midst of a task that might cause work to be lost if the user does not complete the task; if the changes are published it is willing to terminate the task, accept the context data changes and change its internal state accordingly.</td>
</tr>
<tr>
<td>ACL</td>
<td>Access control lists, which determine the privileges and capabilities a particular user has, are presumed to be maintained by each application.</td>
</tr>
<tr>
<td>Annotation agent</td>
<td>A context agent that whose primary role is to set the data for an annotation data for an annotation subject.</td>
</tr>
<tr>
<td>Annotation data</td>
<td>Additional information from an authentic source that describes an entity or concept identified in a related context subject. Annotation data is represented as data items within an annotation subject.</td>
</tr>
<tr>
<td>Apply</td>
<td>A user choice; the context data changes are applied to all of the applications, including those that indicated that they might loose work performed by the user; this choice is allowed only when there are no busy applications.</td>
</tr>
<tr>
<td>Authentication repository</td>
<td>Enables applications to securely maintain application-specific user authentication data. The repository is used by applications that do not have a built-in means to easily sign-on a user given only a logon name.</td>
</tr>
</tbody>
</table>
AuthenticationRepository (AR) Interface used by applications to securely interact with the repository to store and retrieve user authentication data.

Automatic Log-off Logs the current user off of the User Linked applications on a desktop when the user has not interacted with the applications for an appreciable period of time.

Break Link A user choice; the context changes are applied just to the application with which the user initiated the context changes.

Busy When an application is unable to apply the context change because it is blocked (e.g., it is a single threaded application that has a modal dialog open); these applications are referred to as busy.

Cancel A user choice; when the context change is canceled; the context changes are not published.

CCOW Clinical Context Object Workgroup.

Centralized model In the centralized model of context management, the responsibility for managing the common context is centralized in a common facility that is responsible for coordinating the sharing of the context among the applications.

Chain of trust With the chain of trust, User Link-enabled applications and User Link components work together to ensure that only authorized users are allowed access to a common context system.

Clinical context State information that users establish and modify as they interact with healthcare applications. The context is common because it establishes parameters that should uniformly affect the behavior or operation of multiple healthcare applications.
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common context system</td>
<td>Applications that share the same common context, and have established and maintain a common context link.</td>
</tr>
<tr>
<td>Component architecture diagram</td>
<td>Depicts components and their interfaces, and indicates which interfaces each component use for communicating with other components.</td>
</tr>
<tr>
<td>Component interaction diagram</td>
<td>Illustrates the series of method invocations that components perform on each other in order to implement a particular use case.</td>
</tr>
<tr>
<td>Conditionally accept</td>
<td>When an application might lose work performed by the user if it applies the context changes (e.g., the user was in the process of entering data that would not be applicable in the new context); these applications are referred to as having <em>conditionally accepted</em> the context changes.</td>
</tr>
<tr>
<td>CMA</td>
<td>Context Management Architecture.</td>
</tr>
<tr>
<td>COM</td>
<td>Microsoft’s Component Object Model.</td>
</tr>
<tr>
<td>Component model</td>
<td>The architecture of a system as described in terms of components and the interfaces they must implement in order to be participants in the system.</td>
</tr>
<tr>
<td>Context agent</td>
<td>A service component that from the perspective of an application is a transparent participant in a context change. A context agent’s primary role is to automatically add additional data to the context.</td>
</tr>
<tr>
<td>Context change coupon</td>
<td>Unique identifier that is assigned by the context manager to denote each context change transaction.</td>
</tr>
<tr>
<td>Context changes pending</td>
<td>During the context change survey, the context manager informs each of the applications in the common context system (except for the application that instigated the changes) that there are pending context data changes. Each application decides whether or not it wants to accept the changes. All</td>
</tr>
</tbody>
</table>
applications must accept in order for the context to change.

Context change survey

In the first step of completing a context change transaction the context manager surveys the applications. Each application is informed that there are a candidate set of context data changes and is asked to indicate whether it can accept these changes.

Context change transaction

A multi-step process in that coordinates changes to the common context data. First, an application begins a transaction. The application sets a transaction-specific version of the common context data. Second, the context manager conducts a context change survey. Third, the context manager reports the survey results to the application that began the transaction. Finally, the application indicates whether the changes are to be applied or cancelled. The decision as to how to proceed may involve the user. If changes are applied, then the transaction-specific version of the context data becomes the new context. Otherwise the transaction-specific context data is discarded.

Context manager

Coordinates applications each time there is a context change transaction. It is also the “owner” of the authentic context data for the system.

Context participants

Applications that set and/or get context data from the context manager. They must follow the policies established later in this document in order to behave as proper context management “citizens.”

Context subject

A subject represents a real-world entity or concept that is identified as part of the overall common clinical context.

ContextAgent (CA)

Interface implemented by the context agent and used by a context manager to inform the context agent that the clinical context has changes pending and that the agent should perform its context data mapping responsibilities.
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContextData (CD)</td>
<td>Interface implemented by the context manager; used by applications to set/get the data items that comprise the common context.</td>
</tr>
<tr>
<td>ContextManager (CM)</td>
<td>Interface implemented by the context manager; used by applications to join/leave a common context system and to indicate the start/end of a set of changes to the common context data.</td>
</tr>
<tr>
<td>ContextParticipant (CP)</td>
<td>Interface implemented by an application that wants to participate in a common context system; used by the context manager to inform an application that the context has changed.</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture.</td>
</tr>
<tr>
<td>Corroborating data</td>
<td>Corroborating data can be used by applications and/or users as a basis for checking further that the identified context subject is what was expected.</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed version of Microsoft’s Component Object Model.</td>
</tr>
<tr>
<td>Digital signature</td>
<td>Formed using public key / private key encryption techniques, a digital signatures enables</td>
</tr>
<tr>
<td>Dispose</td>
<td>A component performs an implicit or explicit action, which is technology-specific, when it no longer intends to use a particular reference. The latter action is referred to as disposing an interface reference.</td>
</tr>
<tr>
<td>Distributed model</td>
<td>In the <em>distributed model</em> of context management, the responsibility for managing the common context is uniformly distributed among the applications. There is no central point of common context management.</td>
</tr>
<tr>
<td>Empty context</td>
<td>A context is not defined for any subject, either because no context identifier items are present in the context data (as is the case when a context manager</td>
</tr>
</tbody>
</table>
is first initialized) or because the values of all of the identifier items for the subject that are present in the context data are *null* (as is the case when an application explicitly indicates that the context is empty).

<table>
<thead>
<tr>
<th>Empty context subject</th>
<th>A context subject is <em>empty</em> when a real-world entity or concept is not currently identified. For example, for the patient subject, this means that a patient is not currently identified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier data</td>
<td>Data that identifies a real-world entity or concept (such as a specific patient or a specific encounter). Identity information is required in order to establish a common context between applications that involves a real-world entity or concept. The string “id” indicates identifier data.</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language. IDL can specify: an interface’s symbolic name, the set of component properties and methods that can be accessed via the interface, the name and data type of each property, the names and data types for each method’s input and outputs, and the names and data content for each method’s exceptions.</td>
</tr>
<tr>
<td>Instigator</td>
<td>The application that began the current context change transaction.</td>
</tr>
<tr>
<td>ImplementationInformation (II)</td>
<td>Interface implemented by the context manager and mapping agent; used by applications, context management components, and tools, to obtain details about a component’s implementation, including its revision, when it was installed, etc.</td>
</tr>
<tr>
<td>Interface interrogation</td>
<td>The interfaces that a component implements can be determined by other components at run-time through direct interrogation.</td>
</tr>
<tr>
<td>Interface reference registry</td>
<td>A service that contains references to component interfaces. Components can use the registry to obtain interface references to each other.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Log-off</td>
<td>The termination of a user’s session with an application; it assumed that logging-off does not require user authentication.</td>
</tr>
<tr>
<td>Mapping agent</td>
<td>A context agent that whose primary role is to add additional subject-specific context identifier items to the context data.</td>
</tr>
<tr>
<td>MappingAgent (MA)</td>
<td>Interface implemented by a mapping agent and used by a context manager to inform the mapping agent that the clinical context has changes pending and that the mapping agent should perform its context data mapping responsibilities.</td>
</tr>
<tr>
<td>Message authentication code</td>
<td>A secure hash value produced from a data stream that consists of data that is openly communicated between two parties, and “secret” data that they both know but do not openly communicate.</td>
</tr>
<tr>
<td>Message digest</td>
<td>A digital signature is formed by applying a secure hash function (alternatively known as a one-way hash function) to the data that is to be transmitted. The resulting hash value is referred to as the message digest, as it is a numeric surrogate for the plain-text message.</td>
</tr>
<tr>
<td>Null item value</td>
<td>The value of a context data item can be set to the distinguished value of null to indicate that the item does not have a valid value.</td>
</tr>
<tr>
<td>OMA</td>
<td>Object Management Group’s Object Management Architecture.</td>
</tr>
<tr>
<td>Participant coupon</td>
<td>Unique identifier that is assigned by the context manager to denote each context participant within a system, including applications and mapping agents.</td>
</tr>
<tr>
<td>Passcode</td>
<td>Similar to passwords used by people. However, because passcodes are only used by computer programs, they can be much longer and complex than passwords typically are. This makes passcodes</td>
</tr>
</tbody>
</table>
extremely hard to guess, even when brute force techniques are employed.

<table>
<thead>
<tr>
<th>Patient Link</th>
<th>Enables the user to select the patient of interest once from any application as the means to automatically “tune” all of the applications to the selected patient.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient subject</td>
<td>The context subject of <em>Patient</em> is defined for Patient Link. The context data identifier item for this subject is the patient’s medical record number. The patient’s given name is not used as an identifier.</td>
</tr>
<tr>
<td>Principal interface</td>
<td>Every component implements at least one well-known interface, referred to as the component’s principal interface. The principal interface enables components to perform initial interface interrogations because the name of the principal interface is known a priori, and because all components implement it.</td>
</tr>
<tr>
<td>Private key / Public key</td>
<td>An approach for encrypting data, and for creating digital signatures, wherein a matched set of security keys is used. The private key remains the secret of its owner. The matching public key can be disseminated. X can send a message that only Y can read by encrypting the message using Y’s public key. Y decrypts the message using its private key. Alternatively, Y can digitally sign its messages using its private key. X can validate Y’s signature using Y’s public key.</td>
</tr>
<tr>
<td>Pull-model</td>
<td>A shared component is used to maintain the shared context data. Applications update this resource to change the data. Other applications periodically poll the component to determine if the data has changed.</td>
</tr>
<tr>
<td>Push-model</td>
<td>A shared component is used to maintain the shared context data. This component notifies applications whenever the data is changed. In order to receive a notification, an application must have first explicitly indicated its interest in being notified.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Reauthentication time-out</td>
<td>Requires the currently signed-on user to reauthenticate herself before being allowed to continue using the applications on a clinical desktop. The time-out occurs when the user has not interacted with the desktop for an appreciable period of time.</td>
</tr>
<tr>
<td>Repository</td>
<td>See authentication repository.</td>
</tr>
<tr>
<td>RMI</td>
<td>Java Remote Method Invocation mechanism.</td>
</tr>
<tr>
<td>RSA</td>
<td>A popular public key / private key algorithm.</td>
</tr>
<tr>
<td>Secure (or one-way) hash function</td>
<td>A function used for producing a unique numeric surrogate from an arbitrary data stream. It is improbable that two different data streams will yield the same hash value. A secure hash function is an essential part of the infrastructure needed to support the use of digital signatures.</td>
</tr>
<tr>
<td>SecureBinding (SB)</td>
<td>Interface used by applications to establish a secure communications binding with the context manager before using the SecureContextData interface. Also used by applications to establish a secure communications binding with the authentication repository before using the AuthenticationRepository interface.</td>
</tr>
<tr>
<td>SecureContextData (SD)</td>
<td>Interface similar to the ContextData interface defined for Patient Link; this interface is used by applications to securely set/get the values for the items (logically represented as name-value pairs) that comprise the clinical context.</td>
</tr>
<tr>
<td>S-HTTP</td>
<td>Secure Hyper-Text Transfer Protocol.</td>
</tr>
<tr>
<td>Sign on</td>
<td>The act of identifying oneself to an application, prior to initiating a user session, in a manner that can be authenticated by the application, typically involving a secret password or a biometric reading (such as a thumb-print scan).</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer. SSL enables secure (i.e., encrypted) transmission of data between a client and a server. It also enables a client to authenticate a server (and a server to authenticate a client).</td>
</tr>
<tr>
<td>Stat admission</td>
<td>Occurs when an application needs to enable the user to record information about a patient even if an identifier for the patient is not known.</td>
</tr>
<tr>
<td>Technology neutral</td>
<td>Means that the common clinical context approach should work equally well with any one of a candidate set of relevant technologies.</td>
</tr>
<tr>
<td>Use case diagram</td>
<td>Depicts the actors (human and/or computer-based) and the roles that they play when participating in an interesting scenario.</td>
</tr>
<tr>
<td>Use case interaction diagram</td>
<td>Illustrates the high-level interactions between the actors that participate in the use case.</td>
</tr>
<tr>
<td>User Link</td>
<td>Enables the user to securely logon once to any application as the means to automatically “tune” all of the applications to the user.</td>
</tr>
<tr>
<td>User subject</td>
<td>The context subject of User is defined for User Link. The context data identifier item for this subject is the user’s logon name. The user’s given name is not used as an identifier.</td>
</tr>
<tr>
<td>User Link-enabled application</td>
<td>An application that implements the CMA User Link capability.</td>
</tr>
</tbody>
</table>