Health Level Seven Standard

Context Management Specification
Component Technology Mapping: ActiveX
Version CM-1.0

DOCUMENT ID:	HL7SIGVI_5_2_99
REVISION ID:	May 24, 1999
FILE NAME:	hl7_sigvi_activex_cm_1_0 .doc
SUPERCEDES:	n/a

Contents

Figures

Tables

Preface

This document was prepared by Robert Seliger, Sentillion, Inc., on behalf of Health Level Seven's Special Interest Group for Visual Integration (formerly the Clinical Context Object Workgroup --- CCOW). 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.

1 Introduction

This document specifies the details needed to develop Microsoft ActiveX implementations of applications and components that conform to the HL7 Context Management Architecture (CMA). Using this specification, the resulting applications and service components will be able to communicate with each other per the CMA even if they were independently developed.

The scope of this document is limited to the details pertaining to implementing the CMA-specified application and component interfaces using ActiveX Automation (formerly known as OLE Automation). This sub-technology within the ActiveX portfolio of technologies is supported by a wide range of Microsoft and non-Microsoft development tools.

Visual Basic® 4.0 is used as the "lowest common denominator" baseline programming language for developing context participant applications. The collective capabilities of Visual Basic® 5.0 (as opposed to 4.0) , Visual C++® 5.0, and Visual J++® 1.1 (Microsoft's implementation of Java) are used as the baseline baseline programming language implementations for developing CMA components, including the context manager, patient and user mapping agents, and authentication repository. This specification is also forwards-compatible with more recent versions of these tools.

However, any development tool that supports the creation of Automation clients and servers, and in particular supports the IQueryInterface idiom, should enable the development of applications and components that conform to this specification.

1.1 Assumptions

It is assumed that the reader is familiar with Microsoft's ActiveX technology and with the Microsoft's underlying Component Object Model (COM).

1.2 Compatability

This specification is compatible with the following host operating systems:

- Windows NT Workstation 4.0 service pack 3, or later
- Windows 95 OSR2 or later

This specification is compatible with at least the following programming language implementations:

- Visual C++ 5.0 or later

- Visual Basic 4.0 or later
- Visual J++ 1.1 or later with Microsoft's Java SDK 3.1 or later and Microsoft's Java Virtual Machine 5.00.3161 or later

The specification is likely to be compatible with other implementations of these languages, as well as with other programming languages.

1.3 Technology Mapping

The HL7 Context Management Architecture specification is technology-neutral. This means that while an underlying component system is assumed, a specific system is not identified within the architecture. It is the purpose of this document, and its companions for other component technologies, to map the CMA to a specific target technology. For Automation, the technology-specific details specified in this document include (but are not limited to):

- multiple interfaces
- interface reference management
- dual interface requirements
- registry settings
- ActiveX Java wrappers for ActiveX components
- error handling
- implementable interface definitions

It is beyond the scope of this document to provide all of the details that are needed in order to fully implement conformant CMA applications and components. The necessary additional details are covered in a series of companion specification documents, starting most notably with the Health Level Seven Context Management Specification, Technology- And Subject-Independent Component Architecture, CM-1.0.

As illustrated in Figure 1, 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.

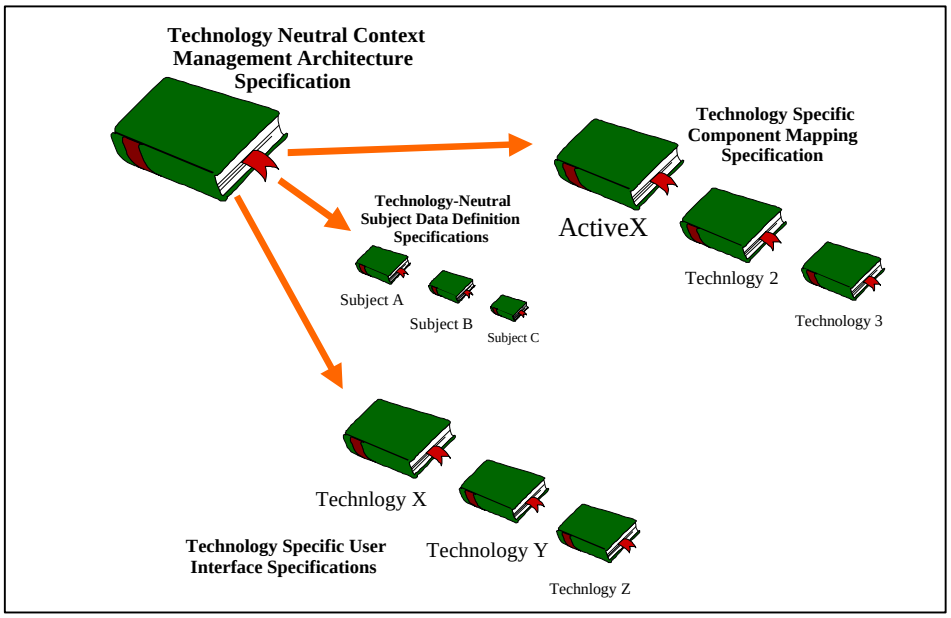

Figure 1: Organization of HL7 Context Management Specification Documents

The context management subjects and technologies that are of interest are determined by the HL7 constituency:

- There is an HL7 context management data definition specification document for each of the standard link subjects. Each document defines the data elements that comprise a link subject. Concurrent with the publication of this document, the following documents have been developed:

  Health Level-Seven Standard Context Management Specification Data Definition: Patient Subject, Version CM-1.0

  Health Level-Seven Standard Context Management Specification, Data Definition: User Subject, Version CM-1.0

- 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 OS, Version CM-1.0

Finally, there is an HL7 context management component technology mapping specification document for each of the component technologies. Each document provides the technology-specific details needed to implement CMA-compliant applications and the associated CMA components, as specified in this document. This document serves the role of specifying the details for a CMA implementation using Microsoft's ActiveX technology.

2 Component Model Mapping

Each interface defined in the CMA specification is implemented as an ActiveX automation interface. All of the components defined in the CMA specification, including context participant applications, are clients as well as servers. In the parlance of ActiveX, they are all Automation clients and servers because they implement and use Automation interfaces.

Context participant applications are only currently required to implement a single Automation interface. However, context managers and mapping agents are required to implement multiple distinct Automation interfaces.

The COM IUnknown::QueryInterface idiom is used to enable context components to acquire each others' interface references through interface interrogation. (Note that Visual Basic implements IUnknown::QueryInterface "under the covers" via the Visual Basic assignment operator.) The COM interface IUnknown serves as a context component's principal interface. See the chapter *Component Model* in the document HL7 Context Management Specification, Technology- And Subject- Independent Component Architecture, CM-1.0 for a discussion about interface interrogation and principal interfaces.

In some cases a component obtains a reference to IQueryInterface for another component from the Windows registry. This registry serves as the interface reference registry described in the chapter *Component Model* in the document HL7 Context Management Specification, Technology- And Subject- Independent Component Architecture, CM-1.0. In other cases, components pass interface references to each other as method parameters.

The various Automation interfaces employed in a common context system are shown in Figure 1. The means by which the various CMA compliant applications and components obtain interface references to each other are described in Table 1.

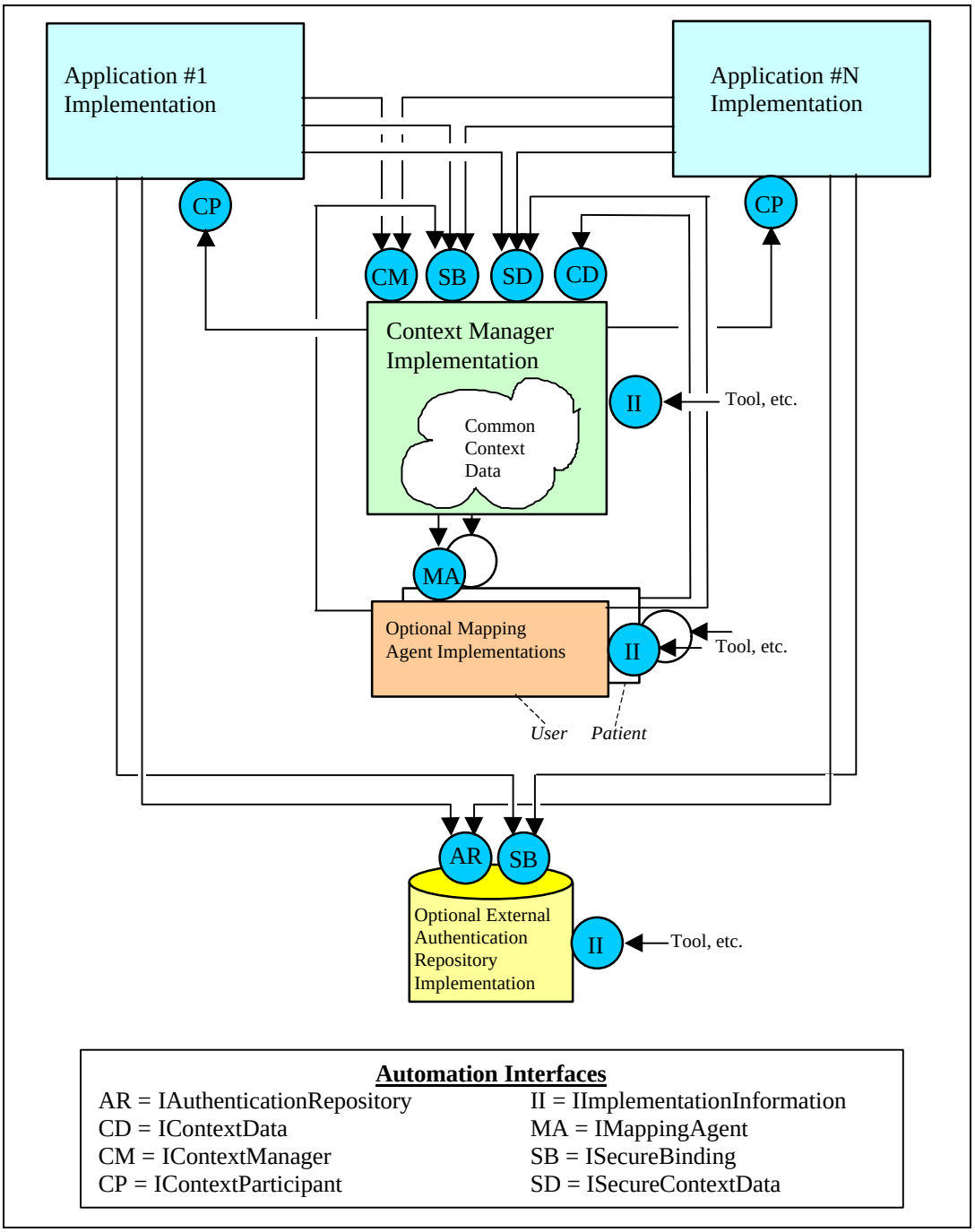

Figure 2: Automation Interfaces in a Common Context System

Automation Server	**Client's means for obtaining server's interface reference(s) …**	
	Automation Client	**Means for obtaining reference**
Context Manager's IContextManager and IContextData interfaces.	Context Participant	A context participant obtains a reference to the context manager's IUnknown interface from the Windows registry. The context participant then performs IUnknown::QueryInterface on the context manager to get the desired interface references.
Context Manager's IContextData interface.	Mapping Agent	The context manager provides a reference to its IUnknown interface to the mapping agent when the context manager calls IMappingAgent::ContextChangesPending. The mapping agent then performs IUnknown::QueryInterface on the context manager to get the desired interface reference.
Mapping Agent's IMappingAgent and IImplementationInformation interfaces.	Context Manager	The context manager obtains a reference to the mapping agent's IUnknown interface from the Windows registry. The context manager then performs IUnknown::QueryInterface on the mapping agent to get the desired interface references.
Context Participant's IContextParticipant interface.	Context Manager	A context participant provides a reference to its IContextParticipant interface to the context manager when the context participant calls IContextManager::JoinCommonContext.
Authentication Repository's IAuthenticationRepository	Context Participant	A context participant obtains a reference to the authentication repository's IUnknown interface from the Windows registry. The context participant then performs IUnknown::QueryInterface on the authentication repository to get the desired interface references.

Table 1: How Interface References Are Obtained

3 Interface Reference Management

In order to “possess” an interface reference, as described in the chapter *Component Model* in the HL7 Context Management Specification, Technology- And Subject- Independent Component Architecture, CM-1.0 document, COM interface reference counts should be incremented and decremented in accordance with COM conventions. In general, a component performs IUnknown::AddRef to “possess” an interface reference. Conversely, a component performs IUnknown::Release to “dispose” an interface reference.

4 Dual Interfaces

Dual Interfaces are a COM optimization that enables an Automation interface to be called using a run-time dispatching mechanism (i.e., so called *dispatch interfaces*), or directly via a compile-time binding mechanism (i.e., so called v-table interfaces). The latter approach generally results in better performance. Dual interfaces accommodate the widest possible range of application development tools, from interpreted late binding languages like Smalltalk and VisualBasic to compiled early binding languages like C and C++.

Context manager, mapping agent, and authentication repository implementations shall expose their CMA-defined Automation interfaces as dual interfaces. This may limit the choice of programming language for these components to just those that support the development of dual interfaces. However, the advantage is better overall run-time performance.

Context participant applications can choose to implement their CMA-defined IContextParticipant interface as a dispatch interface or as a dual interface. This enables application developers to use a wide range of programming languages, as not all languages support dual interfaces.

5 Windows Registry Settings

ActiveX components can have a wide variety of Windows registry entries. It is not unusual for these entries to become quite complex. An objective of this document is to specify the simplest registry entries that will enable applications and components that conform to the CMA specifications to be implemented using any of the common ActiveX-capable programming languages and still seamlessly interoperate.

The context manager shall be registered in the Windows registry. This enables context participant applications to locate and bind to the context manager. If present, a mapping agent shall also be registered in the Windows registry. This enables the context manager to locate and bind to the mapping agent. Finally, if present, the authentication repository shall be registered in the Windows registry. This enables context participant applications to locate and bind to the authentication repository.

ActiveX component registry entries often include implementation-specific information, such as the file name and path to the component's executable code, and may vary depending upon how the component has been implemented (e.g., executable vs. dynamic link library). However, the registry entry for an ActiveX component can use a program identifier (ProgID), which is a symbolic name for the type of component, as a registry key. A registry key is used to locate a registry entry (known as a *value*).

The value associated with a ProgID is the component's class identifier (CLSID), which denotes an implementation of the component. By fixing the ProgID, it is possible to write client's for a type of component such that the client does not need to know anything about the component's implementation. Instead, the client uses the ProgID to locate the component's CLSID at run-time. The CLSID is then used to create an instance of the component, or to connect to an existing instance of a running component.

In summary, ProgID's are invariant across implementation. Therefore, no matter how they are implemented, all of the CMA compliant applications and components shall use the ProgId's defined below[1]:

- The context manager shall be registered using the ProgID sub-key string, "CCOW.ContextManager". The CLSID under which a context manager is registered shall be different for different context manager implementations.

[1] These ProgID's are the same as defined by the Clinical Context Object Workgroup, upon whose original specification this specification is based.

- The patient mapping agent shall be registered using the ProgID sub-key string, “CCOW.MappingAgent_Patient”. The CLSID under which the patient mapping agent is registered shall be different for different patient mapping agent implementations.

- The user mapping agent shall be registered using the ProgID sub-key string, “CCOW.MappingAgent_Patient”. The CLSID under which the user mapping agent is registered shall be different for different user mapping agent implementations.

- The authentication repository shall be registered using the ProgID sub-key string, “CCOW.AuthenticationRepository”. The CLSID under which the authentication repository is registered shall be different for different authentication repository implementations.

The ProgID prefix “CCOW” is reserved for use by HL7 for creating future CMA-related ProgIDs. A CMA-compliant application or component shall not use this prefix other than as specified in this document.

The use of a common ProgID but implementation-specific CLSID requires additional effort on the part of context manager and mapping agent developers. It may also require additional effort on the part of context participant developers:

- Context manager, mapping agent, and authentication repository implementations shall provide ActiveX Java wrapper classes for their CMA coclasses and interfaces as part of their installation package. The details of how these wrapper classes should be prepared and packaged are described below. These wrapper classes are needed in order hide the ActiveX implementation details of these components, including their CLSIDs, from J++ Automation clients for these components.

- Context manager, mapping agent, and authentication repository implementations shall each provide ActiveX-compliant registry entries in `HKEY_CLASSES_ROOT\Interface\` for each of their CMA-specified Automation interfaces. This information is needed so that the Automation clients for these components can create instances of these interfaces.

- Context manager, mapping agent, and authentication repository implementations shall each provide an ActiveX-compliant registry entry `HKEY_CLASSES_ROOT\TypeLib\` for their respective type libraries. This information is needed so that the Automation clients for these components can create calls to these interfaces using the dispatch mechanism.

- Developers of CMA-compliant context participant applications and components shall use the ProgId, not the CLSID, to bind to any of the CMA-defined components that are registered in the registry. This enables implementations to be changed without affecting interoperability.

- Developers of J++ CMA-compliant context participant applications and components shall use the ActiveX Java wrapper classes provided with the CMA-defined components of which they are clients. This is as opposed to client-generated wrappers, which require that the client have development time (versus run-time) access to the implementation of the wrapped component's type library. This is not only impractical, but introduces the probability that a J++ client would only work with a specific Automation server implementation.

When these rules are followed, context participant applications and CMA components will interoperate independently of each other's implementations.

6 ActiveX Java Wrappers

Context manager and mapping agent implementations must provide ActiveX Java wrapper classes:

- The Java package name "ccow.contextmanager" shall be used for the context manager package.
- The Java package name “ccow.mappingagent_patient” shall be used for the patient mapping agent package.
- The Java package name “ccow.mappingagent_user” shall be used for the user mapping agent package
- The Java package name “ccow.authenticationrepository” shall be used for the authentication repository package.
- The context manager package shall minimally contain the Java wrapper classes ContextManager.class, IContextManager.class, IContextData.class, ISecureContextData.class, ISecureBinding.class, IImplementationInformation.class and IContextParticipant.class.
- Both of the mapping agent packages shall minimally contain the Java wrapper classes MappingAgent.class, IMappingAgent.class, and ImplementationInformation.class.
- The authentication repository package shall minimally contain the Java wrapper classes AuthenticationRepository.class, IAuthenticationRepository.class, ISecureBinding.class, and ImplementationInformation.class.

The wrapper classes hide component implementation details. One specific detail hidden is the CLSID to be used by J++ Automation clients for these objects. In order to hide these details, the wrapper classes must be created with knowledge of the details that they hide, hence the need for them to be provided with each component implementation.

From the perspective of a J++ Automation client, the wrapper classes will look and behave the same across component implementations. The wrapper classes are dynamically loaded by a J++ client whenever it first accesses the corresponding Automation client.

The installation of a new component will simply cause J++ clients to automatically access a different version of a seemingly identical component.

The wrapper classes for the context manager should be packaged as "package ccow.contextmanager" and located in:

```
%windir%\java\trustlib\ccow\contextmanager
```

The wrapper classes for the patient mapping agent should be packaged as "package ccow.mappingagent_patient" and located in:

```
%windir%\java\trustlib\ccow\mappingagent_patient
```

The wrapper classes for the user mapping agent should be packaged as "package ccow.mappingagent_user" and located in:

```
%windir%\java\trustlib\ccow\mappingagent_user
```

The wrapper classes for authentication repository should be packaged as "package ccow.authenticationrepository" and located in:

```
%windir%\java\trustlib\ccow\authenticationrepository
```

Note that `ccow`, `contextmanager`, `mappingagent_patient`, `mappingagent_user`, and `authenticationrepository` are all lower case.

7 Microsoft's CRYPTO32 API

All ActiveX implementations of CMA-compliant applications and components that use the CMA-defined secure interfaces shall use the RSA public key / private key scheme and shall use the MD5 one-way hash algorithm. It is recommended that Microsoft's Cryptography Application Programming Interface (CRYPTO32) be used, and that the Microsoft RSA Base Provider be selected as the cryptographic service provider.

However, a different API and/or cryptographic service provider implementation can be used as long as it employs algorithms and binary data formats that are functionally identical to those employed by the Microsoft RSA Base Provider as accessed via the CRYPTO32 API.

7.1 Secure Binding Properties

The CMA-defined interface ISecureBinding requires that the bindee indicate to the binder various security properties that depend upon the bindee's implementation. The properties that must be indicated, and the allowed value or values for each property, depend upon the underlying implementation technology.

For an ActiveX implementation, the following secure binding property names and values defined in Table 2: Secure Binding Properties shall be used.

Property Name	**Allowed Value**	**Meaning**
`Technology`	`CRYPTO32`	Microsoft CRYPTO32 or equivalent.
`PubKeyScheme`	`RSA_EXPORTABLE`[2]	Exportable version of RSA public key / private key scheme (employs 40 bit keys).
`HashAlgo`	`MD5`	MD5 secure hash algorithm (creates 128 bit hash).

Table 2: Secure Binding Properties

[2] Public key / private key schemes are subject to United States export restrictions. Specifically, The U.S. Government limits the size (in bits) of the security keys that can be used as part of applications exported by U.S. vendors. The Microsoft Base Service Provider has been approved for export by the U.S. Government. Applications that use this CSP via the CRYPTO32 API should not require additional export approvals.

Property names are not case sensitive. Property values shall be character-encoded per the convention stated in the CMA specification.

7.2 Cryptographic Service Provider

The CRYPTO32 API enables applications to select from a set of cryptographic service providers (CSP). Each CSP provides cryptographic services that can be accessed via the CRYPTO32 API. For CMA-compliant applications and components that are implemented using the CRYPTO32 API, the Microsoft RSA Base Provider shall be used as the cryptographic service provider. The means that the value of the *dwProvType* to the CRYPTO32 function CryptAcquireContext shall be `PROV_RSA_FULL`.

7.3 Creating Digital Signatures

The CRYPTO32 function CryptSignHash is used to create a digital signature. The function CryptVerifySignature is used to verify a signature. Both of these functions accept an optional pointer to a character string for the parameter *sDescription*. The value of this parameter shall be NULL for all calls to these functions as it pertains to creating or comparing signatures used to implement User Link.

7.4 Signature Format

Digital signatures passed via any of the CMA-defined Automation interfaces shall be represented as a string. This string contains binary data that has been character-encoded per the convention defined in CMA specification. The binary data from which a signature string is created is the byte array produced by CryptSignHash. This string must be converted back to binary data in order to be used as an input to CryptVerifySignature.

7.5 Public Key Format

Public keys passed via any of the CMA-defined Automation interfaces shall be represented as a string. This string contains binary data that has been character-encoded per the convention defined in CMA specification. The binary data from which a public key is created is the byte array produced by CryptExportKey with the parameter *dwBlobType* set to PUBLICKEYBLOB. This string must be converted back to binary data in order to be used as an input to CryptImportKey.

7.6 Hash Value Format

Hash values passed via any of the CMA-defined Automation interfaces shall be represented as a string. This string contains binary data that has been character-encoded per the convention defined in CMA specification. The binary data from which a hash value is created is the byte array produced by CryptGetHashParam. Hash values shall be compared for equality by comparing their character-encode string representations. Character case shall not be considered when comparing these strings.

7.7 Key Containers

With CRYPTO32, public keys and public key / private key pairs are maintained in key containers. These containers can be created and deleted using the CRYPTO32 API function CryptAcquireContext. Keys can be imported into a container, or keys can be directly generated within an empty container.

7.7.1 Required Containers

An application shall maintain the following key containers:

- A key container for holding its own public key / private key pair.
- A key container for holding the context manager's public key.
- Optionally, a key container for holding the authentication repository's public key.

The context manager shall maintain the following key containers:

- A key container for holding its own public key / private key.
- A key container for holding each designated application's public key.
- A key container for holding the user mapping agent's public key.

The user mapping agent shall maintain the following key containers:

- A key container for holding its own public key / private key.
- A key container for holding the context manager's public key.

The authentication repository shall maintain the following key containers:

- A key container for holding its own public key / private key.

- A key container for holding the public keys for each of applications that use the repository.

The convention for naming these containers and for managing their creation and deletion are described next.

7.7.2 Key Container Naming Convention

All of the key containers shall have unique names when they are co-resident on the same Windows host. The naming convention is defined in Table 3: Key Container Naming Scheme.

Note that all of the letters in a container's name shall be capitalized. Also note that the portion of a container name shown as *APPLICATION-NAME* is the same string that an application provides to the context manager when it joins the common context system.

7.7.3 Key Container Management

An application, context manager, user mapping agent, and authentication repository shall delete any containers that its has created prior to terminating.

However, an application, context manager, user mapping agent, or authentication repository that terminates prematurely might fail to delete some or all of the containers that it has created. When the failed component is next launched it will not be able to create a new container if a previously created container with the same name still exists. This situation shall be handled as follows: The existing container shall be deleted and a new container created. The necessary keys shall be created and/or imported into the new container.

7.7.4 Key Container Security

CMA-compliant applications and components that maintain key containers shall protect their containers from unauthorized access. This means that only the application or component that created the container should be able to access the container.

If key containers are not protected then they are vulnerable to unintended uses. For example, a rogue application might access the keys within a container created by valid CMA-compliant application as a means to impersonate the application within a context management system.

There are a variety of ways to protect key containers. In order to maximize design flexibility for CMA-compliant applications and components, a particular approach is not defined in this specification.

Container created by ….	**Container purpose …**	**Container name …**
Application	Holding own key pair.	CCOW. *APPLICATION-NAME*.SELF
	Holding context manager's public key.	CCOW. *APPLICATION-NAME*.CM
	Holding authentication repository's public key.	CCOW. *APPLICATION-NAME*.AR
Context Manager	Holding own pair.	CCOW.CM.SELF
	Holding an application's public key.	CCOW.CM. *APPLICATION-NAME*
	Holding user mapping agent's public key.	CCOW.CM.MA_USER
User Mapping Agent	Holding own key pair.	CCOW.MA_USER.SELF
	Holding context manager's public key.	CCOW.MA_USER.CM
Authentication Repository	Holding own key pair.	CCOW.AR.SELF
	Holding an application's public key.	CCOW.AR. *APPLICATION-NAME*

Table 3: Key Container Naming Scheme

8 Error handling

The CMA specifies a set of exceptions that can be raised by CMA components. (Context participant applications do not currently throw exceptions).

ActiveX Automation exceptions are implemented in a two-stage process. First, all Automation and dual interface methods return a 32-bit encoded error value, called an HRESULT, to their caller. Secondly, ActiveX components that support the Microsoft-defined IErrorInfo and ISupportErrorInfo interfaces can provide additional error information to clients when requested. This information includes a textual description of the error and the guid[3] of interface that threw the error.

Each of the CMA-specified exceptions is identified by a distinguished HRESULT. Additionally, the context manager, both mapping agents, and the authentication repository shall support the IErrorInfo and ISupportErrorInfo interfaces. Automation clients for these objects should check the HRESULT after each method invocation to determine if an exception has occurred. Clients may then optionally access additional error information via the server component's IErrorInfo interface.

In the Win32 COM implementation there is at most one error object associated with each logical thread of execution (i.e. a thread can logically span multiple processes on the same or different hosts), and that the error object may be overwritten by a subsequent error. Clients should access IErrorInfo immediately after detecting an exception to insure that the error information they obtain is pertinent.

Visual Basic developers should note that the Visual Basic Err object handles all the IErrorInfo manipulations automatically. In the event that a Visual Basic client encounters an exception, the Visual Basic Err object will contain the exception information.

The list of CMA-defined HRESULTS values is shown in Table 4: Exception Codes.

[3] A guid is a globally unique identifier. Every COM interface definition is denoted by a different guid.

Exception	Hexadecimal value	Explanation
NotImplemented	0x80004001L	Method not implemented. This is the same value as defined for the Win32 `E_NOT_IMPL` HRESULT.
GeneralFailure	0x80004005L	An error was detected or a failure occurred. This is the same value as defined for the Win32 `E_FAIL` HRESULT.
ChangesNotEnded	0x80000201L	Attempt to publish context changes before ending the context change transaction.
InvalidContextCoupon	0x80000203L	A context coupon does not match most recently committed coupon or current transaction coupon.
reserved	0x80000204L	
reserved	0x80000205L	
NameValueCountMismatch	0x80000206L	A name array and its corresponding value array do not have the same number of elements.
NotInTransaction	0x80000207L	Attempt to perform a context management transaction method when a transaction is not in progress.
TransactionInProgress	0x80000209L	Attempt to perform a context management method when a transaction is in progress.
UnknownItemName	0x8000020AL	An item name not known.
UnknownParticipant	0x8000020BL	Participant coupon does not denote a known participant.
TooManyParticipants	0x8000020CL	Attempt to join a context that can't accommodate another participant.
AcceptNotPossible	0x8000020DL	Attempt to publish an “accept” decision but there were participants for which it was not possible to obtain a survey response (e.g., these participants were blocked)
BadItemNameFormat	0x8000020EL	An item name does not conform to format rules.
BadItemType	0x8000020FL	An item data type does not conform to data definition for the item.
BadItemValue	0x80000210L	An item value does not conform to the allowed set of values as defined by the data definition for the item.

Exception	Hexadecimal value	Explanation
InvalidTransaction	0x80000211L	A transaction has been invalidated and aborted because it violates one or more semantic integrity constraints.
UndoNotPossible	0x80000212L	Attempt to undo context changes after the transaction has ended.
ChangesNotPossible	0x80000213L	Attempt to set or delete context data after the transaction has ended.
ChangesNotAllowed	0x80000214L	Mapping agent attempts set or delete a context data item that has been set by the participant that instigated the transaction.
AuthenticationFailed	0x80000215L	A signature could not be authenticated.
SignatureRequired	0x80000216L	A signature is required to perform the method.
UnknownApplication	0x80000217L	An application name is not known.
UnknownConnection	0x80000218L	A connection is not known to the authentication repository.
LogonNotFound	0x80000219L	The desired user logon is not found in the authentication repository.
UnknownDataFormat	0x8000021AL	The format of user authentication data requested could not be found in the authentication repository.
UnknownBindee	0x8000021BL	A security binding coupon does not denote a known bindee.
ImproperKeyFormat	0x8000021CL	A public key is not properly formatted.
BindingRejected	0x8000021DL	The identity of a bindee could not be verified.
ImproperMACFormat	0x8000021EL	A message authentication code is not properly formatted.
UnknownPropertyName	0x8000021FL	A property name is not known.
BadPropertyType	0x80000220L	A property data type does not conform to specification.
BadPropertyValue	0x80000221L	A property data value does not conform to specification.
AlreadyJoinedContext	0x80000222L	The application has already joined the context.

Table 4: Exception Codes

9 Character Set

The Unicode character set shall be used to represent all character strings that are transmitted amongst and between CMA-compliant applications and components. The Unicode character set enables representation of virtually any local character set.

The use of ActiveX Automation, in which character strings are represented by the Automation data type BSTR, provides built-in support for Unicode. This means that an ActiveX implementation of a CMA-compliant applications and components will inherently support Unicode for the character strings that are communicated via the CMA-defined ActiveX Automation interfaces.

10 MIDL Listing

The interfaces defined below are an implementable translation of the abstract interfaces definitions documented in the CMA specification. The following rules were applied to produce the translation:

- The prefix “I” is prepended to the names of each interface, per COM conventions.
- The closest available data types supported by Automation were employed (see table below).
- Outputs are mapped as return values (retval) and in/out parameters. Plain out parameters are not used because they are not easily implemented using Visual Basic 5.0. (Note: the use of in/out parameters requires special attention to proper memory management techniques when implementing context managers or context participants with the C++ programming language.)
- Exceptions names are mapped as HRESULTs. Support for exception data values is optional. If supported, the data values should be mapped to formatted strings and made available through the IErrorInfo interface.
- An interface reference to a component’s principal interface is mapped as an IUnknown pointer. A reference to any other component interface is mapped as an IDispatch pointer.
- Sequences are mapped as safe arrays.
- Abstract data types are mapped to Automation data types as follows:

Abstract Data Type	Automation Data Type
`byte`	`unsigned char`
`short`	`short`
`long`	`long`
`float`	`float`
`double`	`double`
`boolean`	`VARIANT_BOOL`
`string`	`BSTR`
`date`	`DATE`
`type`	`VARTYPE`
`variant`	`VARIANT`

The MIDL that follows must be used by all ActiveX implementations of context managers and context participants. This includes interface and class names, and method signatures.

10.1 Type Libraries

All CMA-compliant Automation server component implementations shall provide a type library that is consistent with the interface definitions specified below. A default interface should not be specified for any of these components. Clients should not assume that an Automation server has a default interface. An explicit call to IUnknown::QueryInterface is necessary to obtain a reference to a specific interface from an Automation server.

10.2 IAuthenticationRepository

```
import "oaidl.idl";
import "ocidl.idl";

[
    object,
    uuid(12B28736-2895-11d2-BD6E-0060B0573ADC),
    dual,
    helpstring("IAuthenticationRepository Interface"),
    pointer_default(unique)
]
interface IAuthenticationRepository : IDispatch
{
      [helpstring(“Establish connection with authentication repository”)]
      HRESULT Connect([in] BSTR applicationName,
                  [out, retval] long *bindingCoupon);

   [helpstring(“Terminate connection with authentication repository”)]
       HRESULT Disconnect([in] long bindingCoupon);

   [helpstring("Set user authentication data for specified logon name")]
      HRESULT SetAuthenticationData([in] coupon,
                  [in] BSTR logonName,
                  [in] BSTR dataFormat,
                  [in] BSTR appSignature);

   [helpstring("Delete user authentication data for specified logon name")]
      HRESULT DeleteAuthenticationData([in] coupon,
                  [in] BSTR logonName,
                  [in] BSTR dataFormat,
                  [in] BSTR appSignature);

   [helpstring("Retrieve user authentication data for specified logon name")]
      HRESULT GetAuthenticationData([in] coupon,
                  [in] BSTR logonName,
                  [in] BSTR dataType,
                  [in] BSTR appSignature,
                  [in, out] BSTR *userData,
                  [out, retval] BSTR *repositorySignature);
};
```

10.3 IContextData

```
import "oaidl.idl";
import "ocidl.idl";

[
    object,
    uuid(2AAE4991-A1FC-11D0-808F-00A0240943E4),
    dual,
    helpstring("IContextData Interface"),
    pointer_default(unique)
]
interface IContextData : IDispatch
{
   [helpstring("get the names of all of the context items")]
      HRESULT GetItemNames([in] long contextCoupon, [out, retval] VARIANT *itemNames);

   [helpstring("delete an item(s) from the set of context items")]
      HRESULT DeleteItems([in] long participantCouppn,
                  [in] VARIANT names,
                  [in] long contextCoupon);

   [helpstring("set the value of one or more context items")]
      HRESULT SetItemValues([in] long participantCoupon,
                  [in] VARIANT itemNames,
                  [in] VARIANT itemValues,
                  [in] long contextCoupon);

   [helpstring("get the value of one or more context items")]
      HRESULT GetItemValues([in] VARIANT names,
                  [in] VARIANT_BOOL onlyChanges,
                  [in] long contextCoupon,
                  [out, retval] VARIANT *itemValues);
    };
```


10.4 IContextManager

```
import "oaidl.idl";
import "ocidl.idl";

[
    object,
    uuid(41126C5E-A069-11D0-808F-00A0240943E4),
    dual,
    helpstring("IContextManager Interface"),
    pointer_default(unique)
]
interface IContextManager : IDispatch
{
    [propget, helpstring("property MostRecentContextCoupon")]
      HRESULT MostRecentContextCoupon([out, retval] long *pVal);

    [helpstring("enables an application to join a common context system")]
      HRESULT JoinCommonContext([in] IDispatch *contextParticipant,
                  [in] BSTR sApplicationTitle,
                  [in] VARIANT_BOOL survey,
                  [in] VARIANT_BOOL wait,
                  [out, retval] long *participantCoupon);

    [helpstring("enables an application to leave a common context system")]
      HRESULT LeaveCommonContext([in] long participantCoupon);

    [helpstring("enables an application to start a context change transaction")]
      HRESULT StartContextChanges([in] long participantCoupon,
                  [out, retval] long *pCoupon);

    [helpstring("enables the application that instigated a context change transaction to
    indicate that it has completed its changes")]
      HRESULT EndContextChanges([in] long contextCoupon,
                   [in, out] VARIANT_BOOL *someBusy,
                   [out, retval] VARIANT *vote);

    [helpstring("enables an application to discard any context data changes that it has
    already made")]
      HRESULT UndoContextChanges([in] long contextCoupon);

   [helpstring("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")]
      HRESULT PublishChangesDecision([in] long contextCoupon,
                    [in] BSTR decision);

    [helpstring("enables an application to indicate that it wants to suspend its active
    participation in a common context system while remaining registered as a
    participant")]
      HRESULT SuspendParticipation([in] long participantCoupon);

   [helpstring("enables an application to indicate that it wants to resume active
    participation in a common context system")]
      HRESULT ResumeParticipation([in] long participantCoupon,
                     [in] VARIANT_BOOL wait );
};
```

10.5 IContextParticipant

```
import "oaidl.idl";
import "ocidl.idl";

[
    object,
    uuid(3E3DD272-998E-11D0-808D-00A0240943E4),
    dual,
    helpstring("IContextParticipant Interface"),
    pointer_default(unique)
]
interface IContextParticipant : IDispatch
{
   [helpstring("informs a participant that a change to the common context data is
    pending")]
      HRESULT ContextChangesPending([in] long contextCoupon,
                  [in, out] BSTR* reason,
                  [out, retval] BSTR *returnValue);

   [helpstring("informs a participant that the common context data has changed")]
      HRESULT ContextChangesAccepted([in] long contextCoupon);

   [helpstring("informs a particpant that a context change transaction has been rejected
    by one or more of the other participating applications")]
      HRESULT ContextChangesCanceled([in] long contextCoupon);

   [helpstring("informs a participant that the system is being terminated")]
      HRESULT CommonContextTerminated(void);

   [helpstring("used to test if the participant is alive")]
      HRESULT Ping(void);
};
```


10.6 IImplementationInformation

```
import "oaidl.idl";
import "ocidl.idl";

[
        object,
        uuid(41123600-6CE1-11d1-AB3F-E892F5000000),
        dual,
        helpstring("IImplementationInformation Interface"),
        pointer_default(unique)
]
interface IImplementationInformation : Idispatch
{
    [propget, helpstring("property ComponentName")]
         HRESULT ComponentName([out, retval] BSTR *pVal);

    [propget, helpstring("property RevMajorNum")]
         HRESULT RevMajorNum([out, retval] BSTR *pVal);

    [propget, helpstring("property RevMinorNum")]
         HRESULT RevMinorNum([out, retval] BSTR *pVal);

    [propget, helpstring("property PartNumber")]
         HRESULT PartNumber([out, retval] BSTR *pVal);

    [propget, helpstring("property Manufacturer")]
         HRESULT Manufacturer([out, retval] BSTR *pVal);

    [propget, helpstring("property TargetOS")]
         HRESULT TargetOS([out, retval] BSTR *pVal);

    [propget, helpstring("property TargetOSRev")]
         HRESULT TargetOSRev([out, retval] BSTR *pVal);

    [propget, helpstring("property WhenInstalled")]
         HRESULT WhenInstalled([out, retval] BSTR *pVal);
};
```


10.7 IMappingAgent

```
import "oaidl.idl";
import "ocidl.idl";

[
        object,
        uuid(753D98C0-6CE1-11d1-AB3F-E892F5000000),
        dual,
        helpstring("IMappingAgent Interface"),
        pointer_default(unique)
]
interface IMappingAgent : Idispatch
{
    [helpstring("informs a mapping that a change to the common context data ready for
    mapping")]
        HRESULT ContextChangesPending([in] long mappingAgentCoupon,
                    [in] IUnnknown *contextMgr,
                    [in] long contextCoupon,
                    [in, out] BSTR* reason,
                    [out, retval] BSTR *returnValue);

    [helpstring("used to let Context Manager mapping agent is alive")]
        HRESULT Ping(void);
};
```


10.8 ISecureBinding

```
import "oaidl.idl";
import "ocidl.idl";

[
    object,
    uuid(F933331D-91C6-11D2-AB9F-4471FBC00000),
    dual,
    helpstring("ISecureBinding Interface"),
    pointer_default(unique)
]
interface ISecureBinding : IDispatch
{
    [helpstring("Initiate secure binding")]
       HRESULT InitiatlizeBinding([in] long bindeeCoupon,
                  [in] VARIANT propertyNames,
                  [in] VARIANT propertyValues,
                  [in,out] BSTR *binderPublicKey,
                  [out, retval] BSTR *mac);

    [helpstring("Finalize secure binding")]
       HRESULT FinalizeBinding([in] long bindeeCoupon,
                  [in] BSTR bindeePublicKey,
                  [in] BSTR mac,
                  [out, retval] VARIANT privileges);
};
```

10.9 ISecureContextData

```
import "oaidl.idl";
import "ocidl.idl";

[
    object,
    uuid(6F530680-BC14-11D1-90B1-76C60D000000),
    dual,
    helpstring("ISecureContextData Interface"),
    pointer_default(unique)
]
interface ISecureContextData : IDispatch
{
   [helpstring("return collection of the names in the context")]
      HRESULT GetItemNames([in] long contextCoupon,
                  [out, retval] VARIANT *itemNames);

   [helpstring("set the value of one or more context items")]
      HRESULT SetItemValues([in] long participantCoupon,
                  [in] VARIANT itemNames,
                  [in] VARIANT itemValues,
                  [in] long contextCoupon,
                  [in] BSTR appSignature);

   [helpstring("obtain the value of one or more context items")]
      HRESULT GetItemValues([in] long participantCoupon,
                  [in] VARIANT names,
                  [in] VARIANT_BOOL onlyChanges,
                  [in] long contextCoupon,
                  [in] BSTR appSignature,
                  [in, out] BSTR *managerSignature,
                  [out, retval] VARIANT *itemValues);
};
```


Index

