$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

The OpenEAI Message Protocol
Version 1.0

January, 2003

by

Tod Jackson (tod@openeai.org)
Steve Wheat (steve@openeai.org)

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

Contents

Introduction

Work on the OpenEAI Message Protocol began in December 2000 as a collaborative effort between the University of Illinois and SCT Corp. The intent of this collaboration was to provide a format for quantifying enterprise data objects and to prescribe a protocol to communicate or request changes in the state of these enterprise data objects at their authoritative source. The work of the Open Application Group (http://www.OpenApplications.org) provided initial inspiration for early message structure, and that influence is evident in OpenEAI message structure and message definition practices. The OpenEAI Message Protocol strives to express all actions that can be performed on or for enterprise data objects in both request/reply and publish/subscribe messaging models. This protocol is maintained by the OpenEAI Project (info@OpenEAI.org). The original draft of this document was written by Tod Jackson (tod@openeai.org) and Steve Wheat (steve@openeai.org) in 2002 and published January, 2003.

If you are familiar with the Simple Object Access Protocol (SOAP) and related web services technologies, much of what you read here will be familiar—up to a point, particularly the concepts of defining enterprise data as XML objects (what OpenEAI calls “enterprise message objects”), putting them into messages, and transporting them via your preferred transport mechanism. From there, OpenEAI goes on to specify a general protocol for what you can do with any message object within a message, based on the OpenEAI Project’s documented theories of enterprise application integration and the experience of many messaging implementations by the projects participants.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

The message format specified by the OpenEAI Message Protocol includes administrative information required to successfully support many complex request/reply and publish/subscribe interfaces that have special performance, security, routing, logging, and auditing requirements. While the OpenEAI Message Protocol can be used for linking two or three applications (and it is easy to use for that purpose), it is important to keep in mind that many of the details of OpenEAI address integration requirements within large enterprises that are complex, with many applications to integrate.

You may be familiar with the concept of foundational APIs that help developers by implementing much of the lower-level mechanics of, for example, building XML messages from data objects. OpenEAI employs this concept. In fact, developing foundational APIs has been the primary focus of the project to date. EAI can involve heavy lifting, such as content-based message routing and data value and format translations. Having a foundation to do this grunt work for you, driven by clear configuration artifacts, makes EAI technology practical to use.

A word of encouragement…if you look at the sample messages and other artifacts below and find yourself thinking, "This is complex!" or, "Gee, do I need all this stuff?"…we advise you to suspend your concern: there are solid foundational APIs to much of the work. For example, one of the message objects described in subsequent sections is BasicPerson. Here is how you would query a remote application for a BasicPerson, and receive the reply:

```
import org.openeai.jms.producer.PointToPointProducer;
import com.any_erp_vendor.moa.jmsobjects.person.v1_0.BasicPerson;
import com.any_erp_vendor.moa.jmsobjects.person.v1_0.LightweightPerson;

// Get a preconfigured LightweightPerson out AppConfig (an OpenEAI
// application foundation component you can read about in the API
// Introduction document). LightweightPerson is the object we must
// use to query for a BasicPerson according to the definition of
// the message com.any-erp-vendor.Person.BasicPerson.Query-Request.

LightweightPerson lPerson =
  (LightweightPerson)getAppConfig().getObject("LightweightPerson.v1_0");

// Set the InstitutionalId, which is an identification number, on the
// LightweightPerson to be that of the BasicPerson we're querying for.
// If the data we're passing is not valid, enterprise-quality data
// as prescribed by the enterprise object documents we've completed
// an exception will be thrown.  You can read more about using enterprise
// object documents to specify valid values, translations, and rules in
// the API Introduction document.
try {
  lPerson.setInstitutionalId("123456789");
}
catch (Exception e) {
  // perform appropriate exception handling…
}

// Get a preconfigured BasicPerson object from AppConfig.
BasicPerson bPerson =
  (BasicPerson)getAppConfig().getObject("BasicPerson.v1_0");

// Get a preconfigured, started PointToPointProducer object from AppConfig
// with which to perform the query action via JMS.
PointToPointProducer p2p =
  (PointToPointProducer)getAppConfig().getObject("PointToPoint");
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
// Perform the query.  The query will return a List of BasicPerson
// objects if any are found that match the LightweightPerson we used
// in the query.  If any errors occur during the query, an
// exception will be thrown.  This includes errors that occur in the
// application that handles the request we are sending.
Java.util.List personList = null;
try {
  personList = bPerson.query(lPerson, p2p);
}
catch (Exception e) {
  // perform appropriate exception handling…
}

// Log the name associated with each BasicPerson object returned, so
// we can se something about the results we got back from the remote
// application we queried.
For (int i=0; i<personList.size(); i++) {
  (BasicPerson)aBasicPerson = (BasicPerson)personList.get(i);
  logger.info("Name: " + aBasicPerson.getName().getLastName() + ", " +
aBasicPerson.getName().getFirstName());
}
```

This sends a request and receives the following reply over your message transport.

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE com.any-erp-vendor.Person.BasicPerson.Query SYSTEM
"http://xml.openeai.org/message/releases/com/any-erp-vendor/Person/BasicPerson/1.0/dtd/Query-Request.dtd">
<com.any-erp-vendor.Person.BasicPerson.Query>
    <ControlAreaRequest messageCategory="com.any-erp-vendor.Person" messageObject="BasicPerson"
messageAction="Query" messageRelease="1.0" messagePriority="9" messageType="Request">
        <Sender>
            <MessageId>
                <SenderAppId>edu.uillinois.uihr.NessieApplication</SenderAppId>
                <ProducerId>33b054ea-33a2-4fc0-923b-c62fe2837963</ProducerId>
                <MessageSeq>86</MessageSeq>
            </MessageId>
            <Authentication>
                <AuthUserId>nessie</AuthUserId>
            </Authentication>
        </Sender>
        <Datetime>
            <Year>2001</Year>
            <Month>03</Month>
            <Day>23</Day>
            <Hour>13</Hour>
            <Minute>47</Minute>
            <Second>30</Second>
            <SubSecond>472</SubSecond>
            <Timezone>6:00-GMT</Timezone>
        </Datetime>
        <ExpectedReplyFormat messageCategory="com.any-erp-vendor.Person"
messageObject="BasicPerson" messageRelease="1.0" messageAction="Provide" messageType="Reply"/>
    </ControlAreaRequest>
    <DataArea>
        <LightweightPerson>
            <InstitutionalId>123456789</InstitutionalId>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
        </LightweightPerson>
    </DataArea>
</com.any-erp-vendor.Person.BasicPerson.Query>
```

Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE com.any-erp-vendor.Person.BasicPerson.Provide SYSTEM
"http://xml.openeai.org/message/releases/com/any-erp-vendor/Person/BasicPerson/1.0/dtd/Provide-Reply.dtd">
<com.any-erp-vendor.Person.BasicPerson.Provide>
    <ControlAreaReply messageCategory="com.any-erp-vendor.Person" messageObject="BasicPerson"
messageAction="Provide" messageRelease="1.0" messagePriority="9" messageType="Reply">
        <Sender>
            <MessageId>
                <SenderAppId>edu.uillinois.aits.Paymaster</SenderAppId>
                <ProducerId/>
                <MessageSeq/>
            </MessageId>
            <Authentication>
                <AuthUserId>paymaster</AuthUserId>
            </Authentication>
        </Sender>
        <Datetime>
            <Year>2001</Year>
            <Month>03</Month>
            <Day>23</Day>
            <Hour>13</Hour>
            <Minute>47</Minute>
            <Second>30</Second>
            <SubSecond>673</SubSecond>
            <Timezone>6:00-GMT</Timezone>
        </Datetime>
        <Result action="Query" status="success">
            <ProcessedMessageId>
                <SenderAppId>edu.uillinois.uihr.NessieApplication</SenderAppId>
                <ProducerId>33b054ea-33a2-4fc0-923b-c62fe2837963</ProducerId>
                <MessageSeq>86</MessageSeq>
            </ProcessedMessageId>
        </Result>
    </ControlAreaReply>
    <DataArea>
        <BasicPerson citizen="Yes" citizenshipType="Citizen" deceased="No" disabledVeteran="No"
maritalStatus="Unmarried" vietnamService="No">
            <InstitutionalId>123456789</InstitutionalId>
            <Name>
                <LastName>Thoreau</LastName>
                <FirstName>Henry</FirstName>
                <MiddleName>David</MiddleName>
            </Name>
            <BirthDate>
                <Month>7</Month>
                <Day>12</Day>
                <Year>1817</Year>
            </BirthDate>
            <Gender>Male</Gender>
            <DeceasedDate>
                <Month>5</Month>
                <Day>6</Day>
                <Year>1862</Year>
            </DeceasedDate>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
                <Ethnicity>Caucasian</Ethnicity>
                <Address type="Campus Primary">
                    <Street1>50 Gerty Drive</Street1>
                    <CityOrLocality>Champaign</CityOrLocality>
                    <StateOrProvince>Illinois</StateOrProvince>
                    <Nation>USA</Nation>
                    <ZipOrPostalCode>61820</ZipOrPostalCode>
                    <EffectiveDate>
                        <Month>8</Month>
                        <Day>17</Day>
                        <Year>1835</Year>
                    </EffectiveDate>
                </Address>
                <Address type="Permanent">
                    <Street1>5 Transcendental Pike</Street1>
                    <CityOrLocality>Concord</CityOrLocality>
                    <StateOrProvince>Massachusetts</StateOrProvince>
                    <Nation>USA</Nation>
                    <ZipOrPostalCode>01742</ZipOrPostalCode>
                    <EffectiveDate>
                        <Month>2</Month>
                        <Day>14</Day>
                        <Year>1835</Year>
                    </EffectiveDate>
                </Address>
                <Phone type="Campus Phone">
                    <CountryCode>1</CountryCode>
                    <PhoneArea>217</PhoneArea>
                    <PhoneNumber>555-1212</PhoneNumber>
                    <PhoneExtension>1234</PhoneExtension>
                </Phone>
                <Email type="UI" status="Active" preferred="Yes">
                    <EmailAddress>dead-transcendentalist@uiuc.edu</EmailAddress>
                </Email>
            </BasicPerson>
        </DataArea>
    </com.any-erp-vendor.Person.BasicPerson.Provide>
```

You can move on to the OpenEAI API Introduction Document after reading this document for details on the foundational APIs and developing OpenEAI integrations using them. Don't stress about how to actually implement what you see in this document. That's addressed in detail in the API Introduction document. Think about how the administrative information that you see in these messages could help you. Think about having all of these messages or messages for highly-sensitive transactions sitting in a message log, viewable and searchable from a web application to meet the requirements of those pesky auditors that always seem to pop up when you least expect them. Think about being able to route a publish/subscribe message differently based on which application published it and when it was sent. Think about being able to allow or deny requests to an application based on which application is making the request, which message object or action is being requested, which user is using the application that is making the request, or any other piece of data in the message or that can be queried for using data within the message.

A general understanding of EAI concepts is required in order to understand the mechanics of the OpenEAI Protocol. The following section is a brief primer on EAI concepts. Then we will talk about some basic naming and jargon used by the OpenEAI Message Protocol. Finally, we'll talk about the messaging behavior prescribed by the protocol. It can be helpful to read this document iteratively. You may wish to read it once, and then read the concepts and naming sections again. It may bring them into sharper

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

focus after you have read about the messaging behavior. They are presented first because an introduction to the concepts and the language are required in order to discuss the messaging behavior.

The Root Concept: Authoritative Source

An authoritative source is the definitive or master source for some unit of quantifiable data in the enterprise. This source is usually implemented as an application or as a database. The following are statements that apply this concept:

1. The Paymaster system is the authoritative source for **BasicPerson** information for employees.

2. The SCT Banner system is the authoritative source for **EmergencyContact** data.

3. Icard (the identity card) system is the authoritative source for **InstitutionalIdentity** data.

This concept of authoritative source raises four questions. Answering these questions is the key practice of Enterprise Application Integration.

The OpenEAI Project provides a concrete methodology, strategies, foundation, and deployment patterns to use as organizations strive to answer these questions.

1. How do you quantify data for which applications are authoritative?
2. How do you expose this quantified data to the rest of the enterprise?
3. How do you transport these messages?
4. How do you produce and consume messages?

For a deeper discussion of the concept of authoritative source, see the OpenEAI Methodology Document. Although the concept of authoritative source is central to OpenEAI, it is important to keep in mind that it is a concept to help organize integrations efficiently. It is not a constraint or a limitation. Use it to your benefit. For example, it is easiest, most clear, and overwhelmingly most common to be able to declare one application in an enterprise to be the authoritative source for a message object (like BasicPerson information) or a qualified message object (like BasicPerson information for employees). Other non-authoritative applications can send request messages to the authoritative application to make changes for objects of that type at the authoritative application. The authoritative application can publish synchronization messages that are routed to non-authoritative applications that just need to know about the state of objects for which the application is authoritative.

However, it is possible that some enterprises may believe that they have two or more applications that are authoritative for the exact same data. In reality, all of these applications may be authoritative for that data, but not all at the exact same moment for the exact same data. In this way, the concept of authoritative source can be used to discover the need for an entirely new messaging infrastructure application—an authoritative source service—and to drive the creation of convergence rules for the rare, but real cases in which multiple applications may need to operate on exactly the same data independently of each other, without messaging with each other, and converging the changes through messaging. For thorough examples of how an enterprise of authoritative sources can be organized, see the OpenEAI Implementation Strategies Document.

For now, let's look briefly at how OpenEAI attempts to answer these four key questions.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

1. **How do you quantify data for which applications are authoritative?**

   OpenEAI quantifies data as **XML Enterprise Objects**. From the examples above BasicPerson, EmergencyContact, and InstitutionalIdentity are examples of these quanta. Actually, these three objects have more precise, fully-qualified names, but we will refer to them simply as BasicPerson, EmergencyContact, and InstitutionalIdentity for now. Here are some examples:

   BasicPerson

```
<BasicPerson citizen="Yes" citizenshipType="Citizen" deceased="No" disabledVeteran="No"
maritalStatus="Unmarried" vietnamService="No">
    <InstitutionalId>123456789</InstitutionalId>
    <Name>
        <LastName>Thoreau</LastName>
        <FirstName>Henry</FirstName>
        <MiddleName>David</MiddleName>
    </Name>
    <BirthDate>
        <Month>7</Month>
        <Day>12</Day>
        <Year>1817</Year>
    </BirthDate>
    <Gender>Male</Gender>
    <DeceasedDate>
        <Month>5</Month>
        <Day>6</Day>
        <Year>1862</Year>
    </DeceasedDate>
    <Ethnicity>Caucasian</Ethnicity>
    <Address type="Campus Primary">
        <Street1>50 Gerty Drive</Street1>
        <CityOrLocality>Champaign</CityOrLocality>
        <StateOrProvince>Illinois</StateOrProvince>
        <Nation>USA</Nation>
        <ZipOrPostalCode>61820</ZipOrPostalCode>
        <EffectiveDate>
            <Month>8</Month>
            <Day>17</Day>
            <Year>1835</Year>
        </EffectiveDate>
    </Address>
    <Address type="Permanent">
        <Street1>5 Transcendental Pike</Street1>
        <CityOrLocality>Concord</CityOrLocality>
        <StateOrProvince>Massachusetts</StateOrProvince>
        <Nation>USA</Nation>
        <ZipOrPostalCode>01742</ZipOrPostalCode>
        <EffectiveDate>
            <Month>2</Month>
            <Day>14</Day>
            <Year>1835</Year>
        </EffectiveDate>
    </Address>
    <Phone type="Campus Phone">
        <CountryCode>1</CountryCode>
        <PhoneArea>217</PhoneArea>
        <PhoneNumber>555-1212</PhoneNumber>
        <PhoneExtension>1234</PhoneExtension>
    </Phone>
    <Email type="UI" status="Active" preferred="Yes">
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
            <EmailAddress>dead-transcendentalist@uiuc.edu</EmailAddress>
        </Email>
    </BasicPerson>
```

EmergencyContact

```
<EmergencyContact ownerId="123456789" priority="1">
    <Name>
        <LastName>Emerson</LastName>
        <FirstName>Ralph</FirstName>
        <MiddleName>Waldo</MiddleName>
    </Name>
    <Phone type="Home">
        <PhoneArea>217</PhoneArea>
        <PhoneNumber>555-1317</PhoneNumber>
    </Phone>
    <Address type="Home">
        <Street1>7 Transcendental Pike</Street1>
        <CityOrLocality>Concord</CityOrLocality>
        <StateOrProvince>Massachusetts</StateOrProvince>
        <Nation>USA</Nation>
        <ZipOrPostalCode>01742</ZipOrPostalCode>
        <EffectiveDate>
            <Month>6</Month>
            <Day>11</Day>
            <Year>1821</Year>
        </EffectiveDate>
    </Address>
</EmergencyContact>
```

InstitutionalIdentity

```
<InstitutionalIdentity>
    <InstitutionalId>987654321</InstitutionalId>
    <UnknownPerson>
        <Name current="Yes">
            <LastName>Mozart</LastName>
            <FirstName>Wolfgang</FirstName>
            <MiddleName>Amadeus</MiddleName>
        </Name>
        <BirthDate>
            <Month>1</Month>
            <Day>27</Day>
            <Year>1756</Year>
        </BirthDate>
        <Gender>Male</Gender>
    </UnknownPerson>
</InstitutionalIdentity>
```

2. How do you expose this quantified data to the rest of the enterprise?

OpenEAI exposes this quantified data to the rest of an enterprise with **messages in XML format using the OpenEAI Message Protocol**. OpenEAI XML messages are constrained with XML Document Type Definitions (DTDs). The OpenEAI Project is in the process of providing support

for compatible XML Schema constraints for messages given the wide adoption of this new constraint. Subsequent releases of the OpenEAI APIs will support the use of XML Schema as a constraint. See the OpenEAI Message Definitions Document for complete details.

The OpenEAI Message Protocol consists, at its highest level, of three message types: **requests**, **replies**, and **synchronization** messages. Requests are made by non-authoritative applications of an authoritative application, replies are provided by authoritative applications to non-authoritative applications, and synchronization messages are broadcast by authoritative applications to all non-authoritative applications in the enterprise that are interested in changes in state of various objects. The details of this protocol are the subject of this document. The integration capabilities of OpenEAI are not limited to applications that use the OpenEAI Message Protocol natively. Consult the OpenEAI Implementation Strategies document for strategies for integrating applications that use other standards-based or proprietary EAI protocols with an OpenEAI enterprise. There you will find details about using message transformation, data value translation, and layout management facilities built into the OpenEAI Message Object API.

3. How do you transport these messages?

The OpenEAI Project, along with progressive segments of the IT industry, opted to implement message transport services with a Java Message Service (JMS) provider. This allows flexibility in that the message transport can be any transport which implements JMS, ranging in features from very basic to feature-rich and ranging in cost from free-of-charge to very expensive. This allows an organization to experiment or begin implementation work with a free- or low-cost solution and replace the message transport layer with a more expensive and feature-rich product as needs grow. Usually, the experience an organization gains in working with a free- or low-cost message transport will help crystallize its understanding of what functionality and administration features a message service should provide and what value-added features it would really like. Most organizations have a difficult time producing a meaningful list of message service requirements without cultivating some experience with these products. When the time comes to replace one JMS provider with another, it is a straightforward and fast undertaking to switch providers.

Another benefit of using JMS providers is the ability to accept upgrades to message transport software. This is far more common than an organization deciding to switch providers entirely. The same JMS specification (and the care that JMS providers take to implement it) that allows an organization to switch providers also allows an organization to very quickly accept upgrades from the same provider. See the OpenEAI Deployment Patterns Document for information on deploying with different JMS providers. The preferred OpenEAI deployment patterns outline how to deploy messaging applications to leverage the portability potential of JMS most fully.

The OpenEAI Application Foundation and OpenEAI Message Object API for making applications message-aware are built on the assumption that a JMS provider is the native transport of the enterprise. However, the integration capabilities of OpenEAI are by no means limited to enterprises and partners committed exclusively to JMS. Consult the OpenEAI Implementation Strategies document for strategies on bridging a JMS enterprise with other message transports where necessary, using messaging infrastructure that is available from the OpenEAI Project or infrastructure that can be readily built using the OpenEAI Application Foundation and OpenEAI Message Object APIs.

4. How do you produce and consume messages?

Making information systems and applications natively message-aware, gating them with a message gateway, or bridging transports and/or protocols with a message relay is the day-to-day

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

work of implementing integrations. The OpenEAI methodology suggests a concrete strategy for performing integration analysis and defining any new XML Enterprise Objects, and identifies the types of messaging applications required to implement an integration. The OpenEAI Project also provides foundation (or software development kit) along with implementation strategies that can be used to make existing applications message-aware or to build completely new message-aware applications. This foundation is the OpenEAI Application Foundation API and the OpenEAI Message Object API.

The OpenEAI Project chose to implement this core foundation in Java because of the wide acceptance and adoption of Java technology and the broad flexibility the platform provides. Consult the OpenEAI API Introduction document for details on what the OpenEAI APIs are and how they are used.

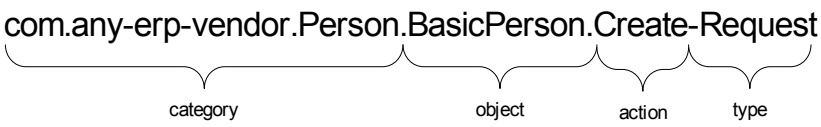
In a nutshell, the OpenEAI methodology defines a process for:

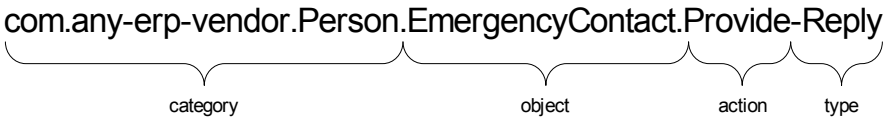
- defining XML Enterprise Objects;
- generating a Message Object API from these XML definitions that give developers a clear business-object-oriented API;
- using OpenEAI application foundation and patterns for gating existing applications using these business objects;
- writing new Java applications using these business objects;
- embedding these business objects in other technologies or bridging to applications that use these objects from other technologies; and
- providing extensive documentation artifacts for enterprise data objects and their movement throughout the enterprise and between trading partners.

The OpenEAI Message Protocol

Message Naming

Messages in the OpenEAI protocol are named by messageCategory, messageObject, messageAction, and messageType. For example,





$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

Let's take a closer look at the first message name above. Here, com.any-erp-vendor.Person is the message category, BasicPerson is the message object, Create is the message action, and Request is the message type. The message category is qualified with the reverse domain name of the organization that defined the message and ends in a category name that describes a logical grouping of objects. For example, this message name indicates that it is in the category com.any-erp-vendor.Person, which means that it was authored by the fictitious ERP vendor who operates in the internet domain any-erp-vendor.com and that this vendor has chosen to categorize some set of messages into the area of Person. Presumably, messages in the Person category contain data about people. The message object is BasicPerson. The name of this object clearly indicates that it contains basic information about a person (see the sample XML for a BasicPerson in the preceding section). The message action is Create and the message type is Request. As we will discuss in more detail below, this indicates that this message is an instruction to create the BasicPerson contained in the message in the application that consumes this request. There are several different ways you can verbalize this name. Most correctly, one would say something like, "this is an Any ERP Vendor person basic person create request." Speaking the name in this way indicates who defined the message, its logical category, object, action, and type. Whenever these message names are written, it is helpful to write them in the format specified above to ensure consistency, clarity, and precision.

The second message named above is also in the com.any-erp-vendor.Person category. However, its message object is EmergencyContact (see the sample XML for an EmergencyContact in the preceding section). The name of this object and its context within the Person category indicates that it contains contact information for a person in case of an emergency. Its message action is Provide and its message type is reply. One would speak this name as an "Any ERP Vendor person emergency contact provide reply." We will discuss in more detail below how the OpenEAI Message Protocol specifies that this is an appropriate reply to a request for emergency contact information (com.any-erp-vendor.Person.EmergencyContact.Query-Request).

The third message listed above is in the package org.any-openeai-enterprise.CoreApplication. This category indicates that it was authored by the fictitious enterprise that operates in the internet domain any-openeai-enterprise.org and that this organization has established a logical category of messages called CoreApplication, which contains messages that are common (or core) to many applications. The message object is InstitutionalIdentity (see the sample XML for an InstitutionalIdentity in the preceding section). The message action is Create and the message type is Sync. The OpenEAI Message Protocol specifies that a generate sync is used to notify other applications about the creation of a new object instance through generation by broadcasting that object's initial state.

Message Categories

In an enterprise, there can be an infinite number of message categories. Categories are indicative of subject areas or areas of operation within an enterprise or within a line of business. Categories are qualified with the reverse domain name of the organization that authored them to distinguish that organization's original message definitions from those of another organization. The concept and value of using domain names to create a global hierarchy is probably familiar and clear from its widespread use in identifying hosts on the internet. The use of reverse domain names to create a clear, global package hierarchy for Java classes is directly parallel to the prescribed use of this convention in the OpenEAI message naming recommendations. This practice allows practitioners of OpenEAI to build a global tree of message definitions, sample

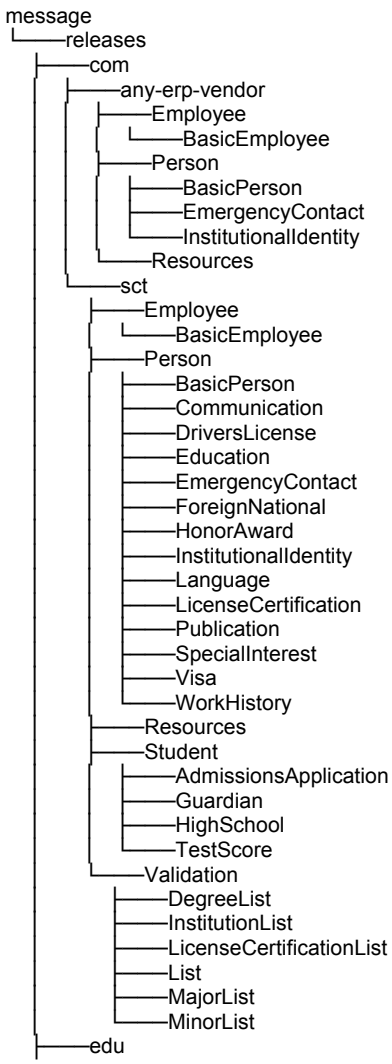
$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

messages, and enterprise object documents to facilitate the sharing of messages and the deployment of messaging artifacts. See the OpenEAI Message Definitions document for more details on the concept of a global message definition tree, sharing and using message definitions within and between organizations, and translating between different message definitions.

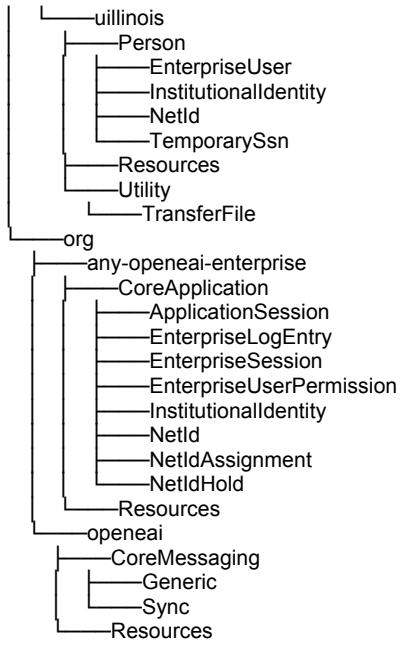
Message Objects

Message objects comprise the business payload of messages. In the examples above, BasicPerson, EmergencyContact, and InstitutionalIdentity are message objects. A message will contain zero or more message objects depending on the type and action of the message as prescribed by the OpenEAI message protocol (see the section on message structure below).

Message objects within the same category should be somehow more related to each other than they are to message objects within another category. The following diagram shows examples of some categories and the message objects within them:



$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$



Again, the relevance of OpenEAI to you has absolutely nothing to do with the specific messages that have already been defined by the OpenEAI Project or any other organization or company. These are some examples of message categories and message objects. OpenEAI prescribes a methodology, message protocol, and foundation with which to implement any integration for any XML Enterprise Object an enterprise can conceive and design. However, one by-product of a clear and uniform process for specifying and organizing XML Enterprise Objects is that is becomes easier for organizations to discuss and when appropriate share and use each other's definitions. This benefit will continue to accrue to all users of OpenEAI concepts and technology over time—a gift that keeps on giving.

Message Actions

There are seven message actions specified for use within the protocol:

1. Create
2. Delete
3. Update
4. Query
5. Provide
6. Generate
7. Error

Unlike message categories and message objects, there is a small, finite number of message actions, because these actions describe fundamental operations that applications can perform with any message object. It is possible that more message actions will be introduced over time; however, such additions would be few and seldom, since these fundamental actions of the protocol have proven adequate for the implementation of many integrations.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

Message Types

There are three message types: **request, reply,** and **synchronization (sync)**. There will probably always only be three message types, because these three types of messages completely cover the two models of messaging that the protocol is intended to address—point-to-point (or request/reply) messaging and publish/subscribe (synchronization) messaging.

While all messaging with a JMS provider or probably any messaging service is technically asynchronous at the transport level, the request/reply messaging specified in the OpenEAI protocol as implemented in its foundation are synchronous. That is, a request is made by a non-authoritative system to an authoritative system, and the thread (or user session) making the request in the non-authoritative system blocks and waits for a reply from the authoritative system. Messages of type request/reply are used to implement synchronous integrations in cases where immediate, confirmed processing is required.

Synchronization messages are used to implement asynchronous, publish/subscribe integrations. Synchronization messages are broadcast from an authoritative system to the rest of the enterprise and consumed by systems that are not authoritative for that data, but need that data (and need to keep it current) to operate. An enterprise can use proprietary or open middleware with varying strategies to route messages from authoritative sources to all interested non-authoritative applications. This process can be very simple or very complex, depending upon the routing, filtering, and security requirements of the enterprise. Consult the OpenEAI Implementation Strategies document for details on how enterprise sync message routing can be performed with OpenEAI messaging infrastructure applications and by other means.

Message Structure

Each message implemented in the OpenEAI Message Protocol has a root element named according to the message category, message object, and message action of the message. For example, the message com.any-erp-vendor.Person.BasicPerson.Create-Request has a root element named com.any-erp-vendor.Person.BasicPerson.Create. This root element has two children: a control area with information about the message required by the OpenEAI Protocol and a data area with the message object data as prescribed by the OpenEAI Message Protocol. Specifically, the root element will have a child control area that is a ControlAreaRequest, ControlAreaReply, or a ControlAreaSync, depending on the type of the message.

For example, a com.any-erp-vendor.Person.BasicPerson.Create-Request message has a root element of com.sct.Person.BasicPerson.Create. That root element has two child elements, ControlAreaRequest and DataArea. A com.any-erp-vendor.Person.EmergencyContact.Provide-Reply has a root element of com.any-erp-vendor.Person.EmergencyContact.Provide. That root element has two child elements, ControlAreaReply and DataArea. An org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create-Sync has a root element named org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create. That root element has two child elements, ControlAreaSync and DataArea.

The control area of each message (either a ControlAreaRequest, ControlAreaReply, or ControlAreaSync) contains detailed administrative information about the message that is used by message-aware applications, message gateways, or messaging infrastructure applications in processing messages. This information is also used by integration administrators to resolve problems or errors in a production environment.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

The following are complete samples of the messages listed above in the message naming section:

com.any-erp-vendor.Person.BasicPerson.Create-Request

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE com.any-erp-vendor.Person.BasicPerson.Create SYSTEM
"http://xml.openeai.org/message/releases/com/any-erp-vendor/Person/BasicPerson/1.0/dtd/Create-Request.dtd">
<com.any-erp-vendor.Person.BasicPerson.Create>
    <ControlAreaRequest messageCategory="com.any-erp-vendor.Person" messageObject="BasicPerson"
messageAction="Create" messageRelease="1.0" messagePriority="9" messageType="Request">
        <Sender>
            <MessageId>
                <SenderAppId>edu.uillinois.uihr.NessieApplication</SenderAppId>
                <ProducerId>4860446b-1e16-4bc0-8d0a-af7675956591</ProducerId>
                <MessageSeq>1024</MessageSeq>
            </MessageId>
            <Authentication>
                <AuthUserId>nessie</AuthUserId>
            </Authentication>
        </Sender>
        <Datetime>
            <Year>2001</Year>
            <Month>3</Month>
            <Day>23</Day>
            <Hour>13</Hour>
            <Minute>47</Minute>
            <Second>30</Second>
            <SubSecond>842</SubSecond>
            <Timezone>6:00-GMT</Timezone>
        </Datetime>
        <ExpectedReplyFormat messageCategory="org.openeai.CoreMessaging" messageObject="Generic"
messageAction="Response" messageRelease="1.0" messageType="Reply"/>
    </ControlAreaRequest>
    <DataArea>
        <NewData>
            <BasicPerson citizen="Yes" citizenshipType="Citizen" deceased="No" disabledVeteran="No"
maritalStatus="Unmarried" vietnamService="No">
                <InstitutionalId>123456789</InstitutionalId>
                <Name>
                    <LastName>Thoreau</LastName>
                    <FirstName>Henry</FirstName>
                    <MiddleName>David</MiddleName>
                </Name>
                <BirthDate>
                    <Month>7</Month>
                    <Day>12</Day>
                    <Year>1817</Year>
                </BirthDate>
                <Gender>Male</Gender>
                <DeceasedDate>
                    <Month>5</Month>
                    <Day>6</Day>
                    <Year>1862</Year>
                </DeceasedDate>
                <Ethnicity>Caucasian</Ethnicity>
                <Address type="Campus Primary">
                    <Street1>50 Gerty Drive</Street1>
                    <CityOrLocality>Champaign</CityOrLocality>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
                        <StateOrProvince>Illinois</StateOrProvince>
                        <Nation>USA</Nation>
                        <ZipOrPostalCode>61820</ZipOrPostalCode>
                        <EffectiveDate>
                            <Month>8</Month>
                            <Day>17</Day>
                            <Year>1835</Year>
                        </EffectiveDate>
                    </Address>
                    <Address type="Permanent">
                        <Street1>5 Transcendental Pike</Street1>
                        <CityOrLocality>Concord</CityOrLocality>
                        <StateOrProvince>Massachusetts</StateOrProvince>
                        <Nation>USA</Nation>
                        <ZipOrPostalCode>01742</ZipOrPostalCode>
                        <EffectiveDate>
                            <Month>2</Month>
                            <Day>14</Day>
                            <Year>1835</Year>
                        </EffectiveDate>
                    </Address>
                    <Phone type="Campus Phone">
                        <CountryCode>1</CountryCode>
                        <PhoneArea>217</PhoneArea>
                        <PhoneNumber>555-1212</PhoneNumber>
                        <PhoneExtension>1234</PhoneExtension>
                    </Phone>
                    <Email type="UI" status="Active" preferred="Yes">
                        <EmailAddress>dead-transcendentalist@uiuc.edu</EmailAddress>
                    </Email>
                </BasicPerson>
            </NewData>
        </DataArea>
    </com.any-erp-vendor.Person.BasicPerson.Create>
```

com.any-erp-vendor.Person.EmergencyContact.Provide-Reply

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE com.any-erp-vendor.Person.EmergencyContact.Provide SYSTEM
"http://xml.openeai.org/message/releases/com/any-erp-vendor/Person/EmergencyContact/1.0/dtd/Provide-Reply.dtd">
<com.any-erp-vendor.Person.EmergencyContact.Provide>
    <ControlAreaReply messageCategory="com.any-erp-vendor.Person" messageObject="EmergencyContact"
messageAction="Provide" messageRelease="1.0" messagePriority="9" messageType="Reply">
        <Sender>
            <MessageId>
                <SenderAppId>edu.uillinois.aits.Banner</SenderAppId>
                <ProducerId/>
                <MessageSeq/>
            </MessageId>
            <Authentication>
                <AuthUserId>banner</AuthUserId>
            </Authentication>
        </Sender>
        <Datetime>
            <Year>2001</Year>
            <Month>3</Month>
            <Day>22</Day>
            <Hour>09</Hour>
            <Minute>34</Minute>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
                <Second>30</Second>
                <SubSecond/>
                <Timezone/>
            </Datetime>
            <Result action="Query" status="success">
                <ProcessedMessageId>
                    <SenderAppId>edu.uillinois.uihr.NessieGateway</SenderAppId>
                    <ProducerId>35cce9e8-f941-4199-89ac-514ea1284597</ProducerId>
                    <MessageSeq>27</MessageSeq>
                </ProcessedMessageId>
            </Result>
        </ControlAreaReply>
        <DataArea>
            <EmergencyContact ownerId="123456789" priority="1">
                <Name>
                    <LastName>Emerson</LastName>
                    <FirstName>Ralph</FirstName>
                    <MiddleName>Waldo</MiddleName>
                </Name>
                <Phone type="Home">
                    <PhoneArea>217</PhoneArea>
                    <PhoneNumber>555-1317</PhoneNumber>
                </Phone>
                <Address type="Home">
                    <Street1>7 Transcendental Pike</Street1>
                    <CityOrLocality>Concord</CityOrLocality>
                    <StateOrProvince>Massachusetts</StateOrProvince>
                    <Nation>USA</Nation>
                    <ZipOrPostalCode>01742</ZipOrPostalCode>
                    <EffectiveDate>
                        <Month>6</Month>
                        <Day>11</Day>
                        <Year>1821</Year>
                    </EffectiveDate>
                </Address>
            </EmergencyContact>
        </DataArea>
    </com.any-erp-vendor.Person.EmergencyContact.Provide>
```

org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create-Sync

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create SYSTEM
"http://xml.openeai.org/message/releases/org/any-openeai-
enterprise/CoreApplication/InstitutionalIdentity/1.0/dtd/Create-Sync.dtd">
<org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create>
    <ControlAreaSync messageCategory="org.any-openeai-enterprise.CoreApplication"
messageObject="InstitutionalIdentity" messageAction="Create" messageRelease="1.0" messagePriority="9"
messageType="Sync">
        <Sender>
            <MessageId>
                <SenderAppId>edu.uillinois.aits.InstitutionalIdentityService</SenderAppId>
                <ProducerId>94c1b2db-121e-4fca-bb23-31f56697c24d</ProducerId>
                <MessageSeq>927</MessageSeq>
            </MessageId>
            <Authentication>
                <AuthUserId>i2s</AuthUserId>
            </Authentication>
        </Sender>
        <Datetime>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
                <Year>2001</Year>
                <Month>3</Month>
                <Day>21</Day>
                <Hour>10</Hour>
                <Minute>34</Minute>
                <Second>45</Second>
                <SubSecond>342</SubSecond>
                <Timezone>GMT -6:00</Timezone>
            </Datetime>
        </ControlAreaSync>
        <DataArea>
            <NewData>
                <InstitutionalIdentity>
                    <InstitutionalId>987654321</InstitutionalId>
                    <UnknownPerson>
                        <Name current="Yes">
                            <LastName>Mozart</LastName>
                            <FirstName>Wolfgang</FirstName>
                            <MiddleName>Amadeus</MiddleName>
                        </Name>
                        <BirthDate>
                            <Month>1</Month>
                            <Day>27</Day>
                            <Year>1756</Year>
                        </BirthDate>
                        <Gender>Male</Gender>
                    </UnknownPerson>
                </InstitutionalIdentity>
            </NewData>
        </DataArea>
    </org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create>
```

The ControlAreaRequest, ControlAreaReply, and ControlAreaSync have the following definitions and usage (from org/openeai/Resources/Segments.dtd). It may be helpful to study these while looking at the sample messages above as concrete examples.

```
<!-- $Revision: 1.5 $
        $Date: 2003/01/06 21:32:44 $
        $Source: /cvs/repositories/openeai/project/message/releases/org/openeai/Resources/1.0/Segments.dtd,v $
 -->
<!-- This file is part of the OpenEAI Application Foundation or
        OpenEAI Message Object API created by Tod Jackson
        (tod@openeai.org) and Steve Wheat (steve@openeai.org).

        Copyright (C) 2002 The OpenEAI Software Foundation

        This library is free software; you can redistribute it and/or
        modify it under the terms of the GNU Lesser General Public
        License as published by the Free Software Foundation; either
        version 2.1 of the License, or (at your option) any later version.

        This library is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
        Lesser General Public License for more details.

        You should have received a copy of the GNU Lesser General Public
        License along with this library; if not, write to the Free Software
        Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
        For specific licensing details and examples of how this software
        can be used to build commercial integration software or to implement
        integrations for your enterprise, visit http://www.OpenEai.org/licensing.
 -->
<!-- Segments.dtd is the file where definitions of all elements with
        children appear.

        The OpenEAI Deployment Patterns suggest that, when using DTDs,
        an enterprise should separate its definitions into four slices
        to improve the management of definitions and to promote the use
        of common object definitions.  The four slices are:

        Resources -     a  file for including definitions from external
                        organizations.  For example, in their own
                        Resources.dtd, an enterprise will reference
                        the OpenEAI definitions, which give them the
                        ControlArea* definitions required by the OpenEAI
                        Message Protocol.

        Segments -      a file where definitions of all elements with
                        children appear

        Fields -        a file where all elements without children appear

        Domains -       a file in which any custom data types are defined

        This approach was learned by observing constraint practices of the
        Open Application Group, and these same practices have served the
        practitioners of OpenEAI well.
 -->
<!-- Release History:

        2002/07/01 1.0 (tod@openeai.org, steve@openeai.org)
 -->
<!-- Include OpenEAI Fields -->
<!ENTITY % ORG.OPENEAI.RESOURCES.FIELDS SYSTEM "Fields.dtd">
%ORG.OPENEAI.RESOURCES.FIELDS;
<!-- Authentication added 01/09/2001 for */ControlArea/Sender/Authentication

        The Authentication element includes authentication information from the
        application that produced the message.

        The AuthUserId element should contain a security principal of the online
        user whose session activity caused the production of the message, if
        appropriate.  Alternately, this element may contain a security principal
        indicative of the trusted application itself that produced the message.

        The optional AuthUserSignature element may contain a digital signature
        or certificate for a user or application indicated in the AuthUserId
        element, if the applications engaged in messaging can support this type
        of authentication, and if such authentication is at all meaningful to
        an enterprise.
 -->
<!ELEMENT Authentication (AuthUserId, AuthUserSignature?)>
<!-- ControlAreaReply added 04/02/2001 for */ControlAreaReply

        The ControlAreaReply element contains metadata for reply messages.
        Here, the message is named with its fully-qualified name in the
        messageCategory, messageObject, messageAction, and messageType
        attributes.  The messageRelease attribute identifies the release
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

level of the message object named in the messageObject attribute. The messagePriority attribute should contain the priority of the message from 0-9, where 0 is the lowest priority and 9 is the highest priority. Messaging foundation and applications may use the value of this attribute to set the priority of the message with the message transport when the message is sent.

The optional SourceInfo element, if present, contains information about the original source of the message, including the SourceId element, which is the name of the source application, as well as the original and appropriate ControlArea* element. In the case of a reply, this may be used when a reply is proxied by a messaging infrastructure application or another messaging application. An example of such an application is the OpenEAI EnterpriseRequestProxy.

The optional TargetInfo element, if present, consists of a TargetId element, which identifies the target application by name. This has not been used for request/reply messaging in the past, but it is used by messaging infrastructure applications that route synchronization messages. The optional TargetInfo element is present in the ControlAreaReply to maintain parallelism with the three ControlArea* elements and because some scenarios for using the TargetInfo element in request/reply messaging can be easily conceived.

The required Sender element consists of a required MessageId and Authentication element. The MessageId element uniquely identifies the message. See the description of the MessageId element for details. The Authentication element provides authentication information for the application producing the message or for the online user of the session of the application producing the message, whichever is more relevant or supported by the applications involved in the integration.

The required Datetime element expresses the precise date and time the message was produced with millisecond precision. See the description of the Datetime element for more details.

The required Result element includes a ProcessedMessageId, which is the MessageId of the processed request to which this is a reply. The ProcessedMessageId can be used to correlate a reply to a request for message transports that may not provide a mechanism to do this more readily. The Result element also includes zero or more Error elements that must contain precise information about any errors encountered in processing the request for which this message is a reply. See the description of the Error element for details.

```
 -->
<!ELEMENT ControlAreaReply (SourceInfo?, TargetInfo?, Sender, Datetime, Result)>
<!ATTLIST ControlAreaReply
    messageCategory CDATA #REQUIRED
    messageObject CDATA #REQUIRED
    messageAction (Provide | Response) #REQUIRED
    messageRelease CDATA #REQUIRED
    messagePriority (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) #REQUIRED
    messageType (Reply) #REQUIRED
>
<!-- ControlAreaRequest added 04/02/2001 for */ControlAreaRequest
```

See the description of ControlAreaReply for a description of the purpose and usage of ControlAreaRequest. These elements are very similar. ControlAreaRequest has the following differences:

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
            The attribute ControlAreaRequest/@messageAction is constrained
            to the list of valid request actions, where
            ControlAreaReply/@messageAction is constrained to the valid list
            of reply actions.

            ControlAreaRequest contains no Result element.  Instead, it
            contains a required ExpectedReplyFormat element.  The
            ExpectedReplyFormat element is an empty element with attributes
            that specify the fully-qualified name of the reply that the
            requesting application expects in response to this request.
-->
<!ELEMENT ControlAreaRequest (SourceInfo?, TargetInfo?, Sender, Datetime, ExpectedReplyFormat)>
<!ATTLIST ControlAreaRequest
    messageCategory CDATA #REQUIRED
    messageObject CDATA #REQUIRED
    messageAction (Assign | Query | Create | Delete | Generate | Update) #REQUIRED
    messageRelease CDATA #REQUIRED
    messagePriority (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) #REQUIRED
    messageType (Request) #REQUIRED
>
<!-- ControlAreaSync added 04/02/2001 for */ControlArea

            See the description of ControlAreaReply for a description of the
            purpose and usage of ControlAreaSync.  These elements are very
            similar.  ControlAreaSync has the following differences:

            The optional SourceInfo element is used frequently in
            synchronization messages by messaging infrastructure applications.
            For example, when a synchronization message is routed or transformed
            by a messaging infrastructure application, that application in
            effect consumes a synchronization message from an authoritative
            source and republishes one or more synchronization messages to
            one or more subscribing applications.  The messaging infrastructure
            application must populate the ControlAreaSync element of the
            SourceInfo element to identify the original source of the message
            and preserve the original ControlAreaSync information from the
            original message, particularly the Datetime of the business event,
            for use in the consumption of the message by the ultimate target
            application.  This information is essential to properly order the
            processing of synchronization messages.  An example of such a
            messaging infrastructure application is the OpenEAI
            EnterpriseTransRouter application for routing enterprise
            messages and translating their values.

            The optional TargetInfo element is similarly used by messaging
            infrastructure applications.  In cases where a messaging infrastructure
            application routes a synchronization message to a specific target
            application, the infrastructure application should populate the
            TargetInfo/TargetId element with the official name of the application
            to which the message is being routed.  This allows other messaging
            infrastructure applications that log all synchronization messages
            to record the target to which each message was published.  An example
            of such a messaging infrastructure application is the OpenEAI
            EnterpriseSyncLogger.

            The required Datetime element must contain the precise date and time
            of the business event that triggered the production of the
            synchronization message.  This is an important distinction between the
            Datetime element in the ControlAreaRequest and the ControlAreaReply
            versus the Datetime in the ControlAreaSync.  This distinction must
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
            be made given the variety of ways that applications can be made to
            publish synchronization messages.  Many application can be made to
            publish synchronization messages within the bounds of the transaction
            that actually changes the data of the message object at the
            authoritative source.  In these cases the Datetime in the
            ControlAreaSync is effectively the precise date and time the sync
            message was published.  However, many applications may record changes
            to enterprise data in the form of business events, which are read
            at some interval to trigger the publication of sync messages.  This is
            a polling strategy.  In these cases, the staged business events MUST
            contain the precise date and time of the business event, so that
            date and time can be used to populate the ControlAreaSync/Datetime
            element.  This information is ABSOLUTELY REQUIRED for applications
            that consume sync messages or for messaging infrastructure applications
            to properly order the processing of synchronization messages.

            The optional Result element is used in a special synchronization message
            in the org.openeai.CoreMessaging category called
            org.openeai.CoreMessaging.Sync.Error-Sync.  As described in the message
            processing behavior specified by the OpenEAI Message Protocol, whenever
            a sync-consuming application encounters a fatal error processing a
            synchronization message, that application MUST publish an
            org.openeai.CoreMessaging.Sync.Error-Sync message that can be consumed
            by messaging infrastructure applications to record the error and alert
            integration administrators.  An example of such a messaging infrastructure
            application is the OpenEAI EnterpriseSyncErrorLogger.  The Result element
            includes the ProcessedMessageId, which is the MessageId of the sync message
            that was being processed when an error or errors were encountered.  The
            optional Result element also contains zero or more Error elements with
            details on the errors encountered consuming the synchronization message.
            See the description of the Error element for more details.
    -->
    <!ELEMENT ControlAreaSync (SourceInfo?, TargetInfo?, Sender, Datetime, Result?)>
    <!ATTLIST ControlAreaSync
        messageCategory CDATA #REQUIRED
        messageObject CDATA #REQUIRED
        messageAction (Assign | Create | Delete | Generate | Update | Error) #REQUIRED
        messageRelease CDATA #REQUIRED
        messagePriority (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9) #REQUIRED
        messageType (Sync) #REQUIRED
    >
    <!-- Datetime added 12/15/2000 for */ControlArea/Datetime

            Within ControlAreaRequest and ControlAreaReply, Datetime is the precise
            date and time the message is produced.  Within a ControlAreaSync,
            Datetime is the precise date and time of the change in state of the message
            object or, in other words, the precise date and time of the business
            event that triggers the sync message.  See the descriptions of
            ControlAreaRequest, ControlAreaReply, and ControlAreaSync for details.

            The prescribed format for Year, Month, Day, Hour, Minute, Second, and
            SubSecond are unpadded numeric values within their appropriate ranges.
            These should be evident for all values except SubSecond, which should
            presently be implemented as a millisecond value between 0 and 999.  The
            value of Timezone should be in the format of offset from universal time.
            These formatting rules are illustrated by the following example.

            <Datetime>
                <Year>2002</Year>
                <Month>1</Month>
                <Day>9</Day>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
                <Hour>9</Hour>
                <Minute>11</Minute>
                <Second>0</Second>
                <SubSecond>475</SubSecond>
                <Timezone>GMT-06:00</Timezone>
            </Datetime>
 -->
<!ELEMENT Datetime (Year, Month, Day, Hour, Minute, Second, SubSecond, Timezone)>
<!-- Error added 12/15/2000 for */ControlArea/Result

        The Error element consists of a type attribute, which identifies
        the error as either a system or application level error.  System
        level errors are more technical or mechanical in nature such as
        an exception as reported by the JDBC driver in connecting to or
        communicating with a database.  An application level error is a
        business logic error of some kind.  The Error element also contains
        an ErrorNumber element with an error number or other appropriate,
        short-form identifier as well as an ErrorDescription element, which
        is a clear, plain-language description of the error in the case of
        and application level error, so that it can be displayed to end users,
        and they will find it meaningful.  In the case of system level errors,
        the description may be more technical such as a Java stack trace or
        other technical trace information.
 -->
<!-- DeleteAction added 01/11/2001 for DataArea/DeleteData

        The DeleteAction element presently contains one attribute called
        'type', which specifies the type of the delete action.  The OpenEAI
        Message Protocol specifies that a consuming application treat a
        delete action of type 'delete' as a standard business transaction,
        applying all appropriate business rules of the application.  In
        other words, a delete action of type 'delete' is a standard delete.
        A delete action of type 'purge' indicates that the object of the
        message be removed completely from the application or truly
        purged.  Support for delete action of type 'purge' can be very
        helpful in support of integration testing.

 -->
<!ELEMENT DeleteAction EMPTY>
<!ATTLIST DeleteAction
    type (Delete | Purge) #REQUIRED
>
<!ELEMENT Error (ErrorNumber, ErrorDescription)>
<!ATTLIST Error
    type (system | application) #REQUIRED
>
<!-- ExpectedReplyFormat added 04/02/2001 for */ControlAreaRequest

        The ExpectedReplyFormat element is an empty element, which consists of
        attributes to specify the fully-qualified name and release number of the
        message object for the reply that the requesting application expects in
        response to its request.
 -->
<!ELEMENT ExpectedReplyFormat EMPTY>
<!ATTLIST ExpectedReplyFormat
    messageCategory CDATA #REQUIRED
    messageObject CDATA #REQUIRED
    messageAction (Provide | Response) #REQUIRED
    messageRelease CDATA #REQUIRED
    messageType (Reply) #REQUIRED
>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
<!-- MessageId added 12/15/2000 for */ControlArea/Sender/MessageId

        The MessageId element uniquely identifies the message.  MessageIds must
        be unique within an enterprise, and it is suggested they be absolutely
        unique (world-wide, forever) to facilitate direct messaging using this
        protocol between trading partners.  Precisely how the MessageId is
        guaranteed to be unique is up to the implementer.  However, as with all
        aspects of the OpenEAI Protocol, the OpenEAI Project provides a concrete
        deployment pattern and foundation that implements this uniqueness that
        implementers may choose to use.

        The MessageId element consists of the SenderAppId, ProducerId, and MessageSeq
        elements.  The SenderAppId element is intended to represent the official name of the
        application that produced the message.  This name should be fully qualified
        with the reverse domain name of your enterprise to help ensure global
        uniqueness of the MessageId.  The ProducerId element is the identifier of the
        component of the application that produced the message.  This could be static
        or dynamic.  For example, the OpenEAI Application Foundation message producers
        can invoke a servlet-exposed UUID generation servlet when they start to generate
        a UUID that is used as the ProducerId.  These message producers can also
        generate this UUID locally.  See the OpenEAI API Introduction document for more
        details on the precise behavior of OpenEAI message producers.  The MessageSeq
        element is the message sequence.  This is usually implemented as a count of
        the number of messages produced by the producer.  Alternately,
        if ProducerId is static, MessageSeq could be implemented as a total count of all
        messages produced by an application.

        The following is an example of a MessageId element from a message published by
        the University of Illinois' Paymaster system:

        <MessageId>
            <SenderAppId>edu.uillinois.aits.Paymaster</SenderAppId>
            <ProducerId>34ce9c65-29f3-4944-ad5b-d77cb356ccbe</ProducerId>
            <MessageSeq>1</MessageSeq>
        </MessageId>
 -->
<!ELEMENT MessageId (SenderAppId, ProducerId, MessageSeq)>
<!-- ProcessedMessageId added 01/12/2001 for PersonEmergencyContactProvide/DataArea/Result

        The ProcessedMessageId is merely a MessageId that appears in
        the Result element of a ControlAreaReply or ControlAreaSync.
        This is only named differently than the MessageId element,
        because experience teaches that administrators frequently confuse
        the MessageId of the processed request or sync with the real MessageId
        of the reply or of the org.openeai.CoreMessaging.Sync.Error-Sync if two
        occurrences of a MessageId element appear as children of different elements in
        a control area.  Therefore, when a MessageId is used in this way,
        it is expressly called a ProcessedMessageId.  See the description
        of the MessageId element for details on what comprises a MessageId.
 -->
<!ELEMENT ProcessedMessageId (SenderAppId, ProducerId, MessageSeq)>
<!-- Result added for PersonEmergencyContactResponse/DataArea/Result on 01/09/2001

        The Result element is used to report errors encountered in processing
        request and sync messages.  The Result element consists of a
        ProcessedMessageId, which identifies the message that was being processed
        when errors were encountered.  See the description of the
        ProcessedMessageId element for details.  The Result element also
        consists of zero or more Error elements.  Each of these consist of an
        ErrorNumber and ErrorDescription element.  See the description of the
        Error element for details.
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
 -->
<!ELEMENT Result (ProcessedMessageId, Error*)>
<!ATTLIST Result
    action (Assign | Query | Create | Delete | Generate | Update) #REQUIRED
    status (success | failure) #REQUIRED
>
<!-- Sender added 12/15/2000 for */ControlArea/Sender

        The Sender element consists of a required MessageId element and a
        required Authentication element, which identify the sending application
        and provide authentication context from the sending application as well as
        uniquely identify the message.  See the description of the MessageId and
        Authentication elements for more details.
 -->
<!ELEMENT Sender (MessageId, Authentication)>
<!-- SourceInfo added 12/15/2000 for */ControlArea/SourceInfo

        The SourceInfo element consists of a required ControlAreaSync,
        ControlAreaRequest, or a ControlAreaReply.  This element is
        used by messaging infrastructure applications, which may route,
        transform, or proxy messages but that still need to retain
        some control information from the application that produced the
        message originally.
 -->
<!ELEMENT SourceInfo (ControlAreaSync | ControlAreaRequest | ControlAreaReply)>
<!-- TargetInfo added 12/15/2000 for */ControlArea/TargetInfo

        The TargetInfo element consists of a required TargetAppName element.
        The TargetAppName element is used by messaging infrastructure applications,
        which may route, transform, or proxy messages to specific target
        applications.  When these specific targets are known, such
        infrastructure applications name the target by using the
        TargetAppName element.
 -->
<!ELEMENT TargetInfo (TargetAppName)>
<!-- TestId added 1/21/2003 for */ControlAreaRequest/Sender/TestId

        The TestId element is used to correlate data specified in a TestSuite with
        message published by a gateway when requests are sent to that gateway
        to perform some action.  The gateway can pull the TestId element out of the
        ControlArea of the request message it consumes and put it on any sync
        messages it publishes.  Then, the OpenEAI Test Suite Application can
        correlate that sync message to a TestSuite-TestSeries-TestCase-TestStep
        listed in the TestSuite document associated to the gateway being tested.
-->
<!ELEMENT TestId (TestSuiteName, TestSeriesNumber, TestCaseNumber, TestStepNumber)>
```

Basic Messaging Behavior

Now let's explore the messaging behavior prescribed by the OpenEAI Message Protocol. You were acquainted with the basic naming, jargon, and message structure in the preceding sections to prepare you for this treatment. The preceding sections are a good reference for understanding the message names and the concept of an authoritative source used here. You may also find it helpful to re-read the preceding sections after you've read this section about the core messaging behavior prescribed by the OpenEAI Message Protocol. This section may bring that preliminary

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

information into sharper focus. Iteration is not only a good tactic for software development; it works for reading as well.

In the following discussion, and asterisk (*) is used as a wildcard to indicate any such message for any message object in any message category. For example, *.Query-Request means ***any*** query request message such as…

- an org.any-openeai-enterprise.CoreMessaging.EnterpriseSession.Query-Request,
- a com.sct.Person.BasicPerson.Query-Request,
- an edu.uillinois.Person.InstitutionalIdentity.Query-Request.

*.Query-Request and *.Provide-Reply

A Query-Request requests that the state of a message object or objects be looked up in a remote application and provided back to the requestor in the form of a Provide-Reply. According to the OpenEAI methodology, you can think of this exchange as a non-authoritative application requesting the state of a message object from the application that is authoritative for this data. A Query-Request contains a query object in its DataArea that is used by the application processing the Query-Request to identify the data that is being requested. The reply for a Query-Request is a Provide-Reply that contains the message object or objects requested in the Query-Request in its DataArea element as prescribed by the definitions of the specific Provide-Reply. If no matching message objects can be identified by the application processing the Query-Request based on the query object contained in the Query-Request, then the Provide-Reply returned to the requesting application will contain an empty DataArea element. This is explicitly not an error condition. This is analogous to a database query returning no matching rows. It is not an error, there is simply no data that matches the specifications of the query. This is reflected in the definition of the Provide-Reply messages. Their definitions specify either “zero or more” or “zero or one” message object elements are expected in the DataArea of a Provide-Reply.

In the event that errors are encountered by the application processing the Query-Request, then that application should return the appropriate Provide-Reply to the requesting application with a status of “failure” in the status attribute of the Result element in the ControlAreaReply along with one or more appropriate errors. Errors can be of type “system” or “application”, representing system level and application level errors. System level errors are more technical in nature and are usually sent with more technical descriptions. For example, the database server used by the authoritative application may be unavailable. When it begins to process a Query-Request, it discovers this, cannot re-establish its database connections, and must reply to the requesting application with an error stating that there was a failure processing the Query-Request. The application processing the Query-Request should probably respond with at least one system level error and one application level error. The Result element in the Provide-Reply that might look like the following:

```
<Result action="Query" status="failure">
   <ProcessedMessageId>
      <SenderAppId>edu.uillinois.NessieApplication</SenderAppId>
      <ProducerId>dc336eca-89c6-4205-9911-e74e5e4df790 </ProducerId>
      <MessageSeq>2</MessageSeq>
   </ProcessedMessageId>
   <Error type="application">
      <ErrorNumber>SCT-1001</ErrorNumber>
      <ErrorDescription>Banner unavailable: unable to connect to Banner
database</ErrorDescription>
   </Error>
```

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

```
      <Error type="system">
         <ErrorNumber>10</ErrorNumber>
         <ErrorDescription>
            SQL Exception: ORA-01017: invalid username/password; logon denied
            java.sql.SQLException: ORA-01017: invalid username/password; logon denied

            at java.lang.Throwable.fillInStackTrace(Native Method)
            at java.lang.Throwable.fillInStackTrace(Compiled Code)
            at java.lang.Throwable.&lt;init&gt;(Compiled Code)
            at java.lang.Exception.&lt;init&gt;(Exception.java:42)
            at java.sql.SQLException.&lt;init&gt;(SQLException.java:43)
            at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:114)
            at oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:208)
            at oracle.jdbc.ttc7.O3log.receive2nd(Compiled Code)
            at oracle.jdbc.ttc7.TTC7Protocol.logon(TTC7Protocol.java:230)
            at oracle.jdbc.driver.OracleConnection.&lt;init&gt;(OracleConnection.java:198)
            at oracle.jdbc.driver.OracleDriver.getConnectionInstance(OracleDriver.java:251)
            at oracle.jdbc.driver.OracleDriver.connect(OracleDriver.java:224)
            at java.sql.DriverManager.getConnection(Compiled Code)
            at java.sql.DriverManager.getConnection(DriverManager.java:137)
            at JDBCTest.main(Compiled Code)
         </ErrorDescription>
      </Error>
   </Result>
```

The system level error states that the database is unavailable and contains a Java stack trace generated by the application that originates from the JDBC driver. This is an example of how system level errors are usually more technical. These errors and descriptions are invaluable to administrators and applications that monitor other messaging applications. However, these errors are frequently of little use to users, so it is always good practice to include at least one application-level error that has a more human-usable error code and message, such as the error stating that the Banner database is unavailable in the example above.

There is always a chance that an application consuming a message will encounter an error before it knows exactly what type of request it is processing. This would occur, for example, when an application has difficulty parsing the message. In these cases, the application processing the request must still respond, because application must make every attempt to reply to all requests. In these cases, the org.openeai.CoreMessaging.Generic.Response-Reply is used as a generic reply message that can contain the status of failure and any appropriate system or application level errors.

Examples (follow links for message text)

Query-Requests and successful replies:

com.any-erp-vendor.Person.BasicPerson.Query-Request
com.any-erp-vendor.Person.BasicPerson.Provide-Reply

org.any-erp-vendor.Person.EmergencyContact.Query-Request
org.any-erp-vendor.Person.EmergencyContact.Provide-Reply

org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Query-Request
org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Provide-Reply

Query-Requests and failure replies:

org.any-erp-vendor.Employee.BasicEmployee.Query-Request

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

org.any-erp-vendor.Employee.BasicEmployee.Provide-Reply

org.any-erp-vendor.Employee.BasicEmployee.Query-Request
org.openeai.CoreMessaging.Generic.Response-Reply

*.Create-Request and org.openeai.CoreMessaging.Generic.Response-Reply

A Create-Request requests that a new object be created in a remote application with the state of the object as indicated in the message. According to the OpenEAI methodology, you can think of this exchange as a non-authoritative application requesting that a new object be created at an authoritative application, with the reply from the authoritative application to the non-authoritative application indicating the success or failure of the create action as the authoritative application attempted to create the object, applying all of its business rules.

A Create-Request contains an object in its DataArea element for the consuming application to attempt to create. The reply to all Create-Request messages is an org.openeai.CoreMessaging.Generic.Response-Reply. This generic message confirms the success or reports the failure of the requested create along with any errors.

In the event that errors are encountered by the application processing the Create-Request, then that application should return an org.openeai.CoreMessaging.Generic.Response-Reply to the requesting application with a status of “failure” in the status attribute of the Result element in the ControlAreaReply along with one or more appropriate errors. Errors can be of type “system” or “application”, representing system level and application level errors. System level errors are more technical in nature and are usually sent with more technical descriptions.

Examples (follow links for message text)

Create-Request and successful reply:

com.any-erp-vendor.Person.BasicPerson.Create-Request
org.openeai.CoreMessaging.Generic.Response-Reply

Create-Requests and failure replies:

org.any-erp-vendor.Person.BasicPerson.Create-Request
org.openeai.CoreMessaging.Generic.Response-Reply

org.any-erp-vendor.Person.EmergencyContact.Create-Request
org.openeai.CoreMessaging.Generic.Response-Reply

*.Update-Request and org.openeai.CoreMessaging.Generic.Response-Reply

An Update-Request requests that the state of a message object or objects be changed in a remote application. According to the OpenEAI methodology, you can think of this exchange as a non-authoritative application requesting that the state of an object be changed at an authoritative application and the reply from the authoritative application to the non-authoritative application indicating the success or failure of the update action as the authoritative application attempted to update the state of the object, applying all of its business rules.

An Update-Request contains a BaselineData element and a NewData element in its DataArea element. The BaselineData element contains the state of the object for which an update is being requested prior to the requested change. The NewData element contains the state of the object for which an update is being requested after the requested change. When the consuming application processes an Update-Request, it must use the state of the object found in the BaselineData element to both locate the object to modify and to perform a baseline comparison. If the object for which an update is requested is found and its state matches the state of the object in the BaselineData element of the Update-Request, the consuming application can safely attempt to process the update. However, if the object for which an update is requested is found and its state does not match the state of the object in the BaselineData element, this means that the state of the object in the consuming, authoritative application has changed since the non-authoritative application queried for the state of the object. In this event, there are two options:

1. **The Common Case**

   The consuming application may not attempt to apply the update, because the baseline is stale. Accepting the Update-Request and successfully applying it would result in the loss of the current, valid state of the object, which the user of the requesting application or the requesting application itself has never seen or considered in making new changes. The consuming application should reply with a "baseline stale" (OpenEAI-1014) error in the org.openeai.CoreMessaging.Generic.Response-Reply. The requesting application can process this error and send a Query-Request to get the current baseline and then continue processing successfully. You may recognize this phenomenon from end-user applications you have used. If you are prompted that the data you are trying to update has been changed and are presented with the new data and an interface that either merges in your changes and asks you to re-approve your changes or an interface that asks you make your changes again, then you've seen this type of optimistic locking strategy in practice.

2. **The Uncommon Case**

   The consuming application may employ convergence rules to merge the requested changes into the current baseline state even though the baseline state of the object in the message is stale. This is a rare case, because organizations' business rules rarely allow for changes to data when the previous state has not been considered. However, if, for example, an organization has defined an unusually coarse-grained business object with only loosely related data in the same object and that object has a high propensity to be updated by multiple people or automated applications at exactly the same moment, it may be worth implementing some basic convergence rules. If these rules do not succeed in acceptably converging the current state and new state of the object, then even in this case the consuming application should be prepared to reply with a "baseline stale" (OpenEAI-1014) error in the org.openeai.CoreMessaging.Generic.Response-Reply as described in the common case above.

Examples (follow links for message text)

Update-Request and successful reply:

com.any-erp-vendor.Person.BasicPerson.Update-Request
org.openeai.CoreMessaging.Generic.Response-Reply

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

Update-Requests and failure replies:

org.any-erp-vendor.Person.BasicPerson.Update-Request
org.openeai.CoreMessaging.Generic.Response-Reply

org.any-erp-vendor.Person.EmergencyContact.Update-Request
org.openeai.CoreMessaging.Generic.Response-Reply

*.Delete-Request and org.openeai.CoreMessaging.Generic.Response-Reply

A Delete-Request contains an object in its DataArea element for a remote application to attempt to delete. According to the OpenEAI methodology, you can think of this exchange as a non-authoritative application requesting that an object be deleted at an authoritative application, with the reply from the authoritative application to the non-authoritative application indicating the success or failure of the delete action as the authoritative application attempted to delete the object, applying all of its business rules.

The DataArea element of a Delete-Request contains a DeleteData element that consists of a DeleteAction element and a message object to be deleted. The type attribute of the DeleteAction element specifies whether the requested delete is of type 'delete' or 'purge.' A DeleteAction of type 'delete' indicates that the application processing this request should attempt to perform a standard delete, applying all of its business rules. Many applications never actually remove objects or records from their stores. They inactivate them or perform some other appropriate business logic. A DeleteAction of type 'purge' is a special case, which directs the application processing the request to actually purge or remove the object or records that comprise it from its store entirely. When implementing message support, you may chose to support or not support Delete-Requests with a DeleteAction of type 'purge'. It may be totally inappropriate for some applications. However, it is very useful for automated setup and tear-down of test cases when testing messaging implementations with automated testing tools such as the OpenEAI TestSuiteApplication or other appropriate tools your organization may have. It is common to implement support for DeleteAction of type 'purge' and make that support configurable, so that it may be toggled on when used in development and test environments where testing may occur and toggled off in a production environment where sensitive records should never be completely purged from a system. If a DeleteAction of type 'purge' is not supported, the consuming application should reply with an "invalid delete action" error (OpenEAI-1008) in the org.openeai.CoreMessaging.Generic.Response-Reply if a Delete-Request with a DeleteAction of type 'purge' is received. If a DeleteAction of type 'purge' is supported, but toggled off, the consuming application should reply with a "purge disabled" error (OpenEAI-1009) in the org.openeai.CoreMessaging.Generic.Response-Reply.

When the consuming application processes a Delete-Request, it must use the state of the object found in the DeleteData element to both locate the object to delete and to perform a baseline comparison. If the object for which a delete is requested is found and its state matches the state of the object in the DeleteData element of the Delete-Request, the consuming application can safely attempt to process the delete. However, if the object for which a delete is requested is found and its state does not match the state of the object in the DeleteData element, this means that the state of the object in the consuming, authoritative application has changed since the non-authoritative application queried for the state of the object. In this event, there are two options:

1. **The Common Case**

The consuming application may not attempt to perform the delete, because the baseline is stale. Accepting the Delete-Request and successfully performing it would result in the loss of the current, valid state of the object, which the user of the requesting application or the requesting application itself has never seen or considered when requesting the delete. The consuming application should reply with a "baseline stale" (OpenEAI-1014) error in the org.openeai.CoreMessaging.Generic.Response-Reply. The requesting application can process this error and send a Query-Request to get the current baseline and then continue processing successfully. You may recognize this phenomenon from end-user applications you have used. If you are prompted that the data you are trying to delete has been changed and are presented with the new data and an interface that asks you to re-approve your delete, then you've seen this type of optimistic locking strategy in practice.

2. The Uncommon Case

The consuming application may employ delete rules to determine if the difference between the current state of the object and the state of the object in the DeleteData element of the Delete-Request is immaterial to the business decision to perform the delete. This is even rarer than the uncommon case outlined above for an Update-Request, because organization's business rules almost never allow for deleting data when its current state has not been considered. If these rules do not succeed in determining that the object may be deleted, then even in this case the consuming application should be prepared to reply with a "baseline stale" (OpenEAI-1014) error in the org.openeai.CoreMessaging.Generic.Response-Reply as described in the common case above.

Examples (follow links for message text)

Delete-Request and successful reply:

com.any-erp-vendor.Person.BasicPerson.Delete-Request
org.openeai.CoreMessaging.Generic.Response-Reply

Delete-Requests and failure replies:

org.any-erp-vendor.Person.BasicPerson.Delete-Request
org.openeai.CoreMessaging.Generic.Response-Reply

org.any-erp-vendor.Person.EmergencyContact.Delete-Request
org.openeai.CoreMessaging.Generic.Response-Reply

*.Generate-Request and *.Response-Reply

A Generate-Request contains an object in its DataArea element for a remote application to use as input for a generation operation. According to the OpenEAI methodology, you can think of this exchange as a non-authoritative application requesting that an object be generated by an authoritative application, using an input object. The reply from the authoritative application to the non-authoritative application should indicate the success or failure of the generate action as the authoritative application attempted to generate the object, applying all of its business rules. The

reply should also contain the object that was generated by processing the request. For this reason, the reply to a Generate-Request is a specific Response-Reply for the message object in question and not a generic reply.

There is always a chance that an application consuming a message will encounter an error before it knows exactly what type of request it is processing. This would occur, for example, when an application had difficulty parsing the message. In these cases, the application processing the request must still respond, because application must make every attempt to reply to all requests. In these cases, the org.openeai.CoreMessaging.Generic.Response-Reply is used as a generic reply message that can contain the status of failure and any appropriate system or application level errors.

This type of exchange is common in a service-oriented architecture. For example, many organizations have an identity card application, directory service, or identification features of its ERP system that serve as the master source for identifying individuals by issuing identification numbers. Applications may message with this ID service to determine what the ID number for an individual or vendor is and request that ID numbers be issues if the person or vendors is new to the organization.

Examples (follow links for message text)

Generate-Request and successful reply:

org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Generate-Request
org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Response-Reply

Generate-Requests and failure replies:

org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Generate-Request
org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Response-Reply

org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Generate-Request
org.openeai.CoreMessaging.Generic.Response-Reply

Synchronization Messages

Synchronization messages are used to broadcast the state of objects from applications that are authoritative for an object to non-authoritative applications which need to be informed of the change in state of that object. Whenever an authoritative application publishes a sync message, the complete message must be logged in its entirety. Whenever a non-authoritative application processes a sync message, there is the possibility that it will encounter an error that prevents it from taking the action prescribed by the sync message. In these cases, a log that contains the message that was unsuccessfully processed is required to remedy the error. This practical reason for maintaining a log of all sync messages published also conforms to the general auditing requirements of many real-world enterprises.

A Practical Note on Enterprise Synchronization Message Logs

Strategies for maintaining an enterprise sync message log and applications for

messaging administrators to view and search sync messages are discussed in the OpenEAI Implementation Strategies Document. Whenever you enable an application to publish a sync message, it must also be capable of publishing an exact copy of that message to a central sync message logging service. There are other strategies for maintaining an enterprise sync message log for small enterprises, such as having a central logging service subscribe to and consume messages from the same end points as each of the applications consuming sync messages. However, once you are beyond a couple dozen applications consuming sync messages, you will likely appreciate the capability to have all applications that publish sync message simply publish a copy to a central enterprise sync logging service that never has to be reconfigured to consume additional messages every time a new application is deployed. So whenever you implement synchronization message support, it should include the ability to publish a copy of each sync message to an enterprise sync log destination. If you use the OpenEAI foundational APIs to implement your sync message publication, the ability to publish a copy of each message to a logging destination is implicitly supported. You just need to configure at runtime whether you would like these log copies published and, if so, to which destination. See the OpenEAI API Introduction Document for more details on this feature.

In order to guarantee the integrity of the synchronized data and the consistency of the non-authoritative applications' data stores with an authoritative application's store, sync messages must be processed and applied to non-authoritative applications in the proper order. The proper order is determined by the time the relevant business transaction occurred in the authoritative application. As discussed in the detailed description of the elements of the OpenEAI control areas, this time is found in the Datetime element of the ControlAreaSync.

Frequently, synchronization messages are processed by intermediary, messaging infrastructure applications such as message routers and relays that route messages or bridge transports prior to being delivered to subscribing applications. In these cases, the OpenEAI Message Protocol prescribes that the original ControlAreaSync must always be preserved intact in the SourceInfo element of the ControlAreaSync so consuming applications are informed of the time of the actual business event that precipitated the synchronization message. In other words, the precise Datetime to be used for ordering the processing of sync messages can be found in each sync message, either in the [root element]/ControlAreaSync/SourceInfo/ControlAreaSync element, if it exists, or in the [root element]/ControlAreaSync element, if the former does not exist in a given message.

The following is an example of a sync message from an authoritative application and how it is multiplexed by a message routing application into multiple messages that are delivered to non-authoritative applications.

Examples (follow links for message text)

Sync message published from an ERP system and consumed by a routing application like the OpenEAI Message Router.

com.any-erp-vendor.Person.BasicPerson.Update-Sync

In a sample enterprise, three applications need to consume BasicPerson sync messages to keep themselves up to date with the authoritative ERP system: a directory service, an identity card application, and a legacy payroll system.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

com.any-erp-vendor.Person.BasicPerson.Update-Sync (routed to the directory service)
com.any-erp-vendor.Person.BasicPerson.Update-Sync (routed to the ID card application)
com.any-erp-vendor.Person.BasicPerson.Update-Sync (routed to the payroll system)

org.openeai.CoreMessaging.Sync.Error-Sync

As outlined above in the discussion of the enterprise synchronization message log, whenever a non-authoritative application processes a sync message, there is a chance that it will encounter an error that prevents it from taking the action prescribed by the sync message. When this occurs, the application processing the sync message should publish an org.openeai.CoreMessaging.Sync.Error-Sync message. These messages are defined to contain errors that describe the problem that the application encountered when processing the sync message. Error-Syncs also contain the message identifier of the sync message that was unsuccessfully processed, so that administrators can correlate the Sync-Error message to a sync message logged in the enterprise synchronization message log.

A Note on Messaging Administration Applications

Sites that presently practice OpenEAI perform routine sync error detection and remediation by running regular queries of their enterprise synchronization error logs and synchronization logs. When necessary, they use the message publication tools of their JMS provider to republish the message they have retrieved from their enterprise synchronization logs.

Clearly, with all of these artifacts in place, very cool messaging administration applications can be developed to present messaging administrators with a list of errors (built from the synchronization error log), allowing them to drill down to the message that was unsuccessfully processed (from the synchronization message log), edit the unsuccessfully processed message (if necessary and appropriate), and republish it to the target application. Such applications and many others are presently the subject of work in the OpenEAI Reference Implementations Department.

Examples (follow links for message text)

Sync message published from a message routing application and consumed by an identification card application.

com.any-erp-vendor.Person.BasicPerson.Update-Sync (routed to the ID card application)

Unfortunately, the identification card application encountered errors processing the message, so it published the following Error-Sync message to an enterprise sync error logging service for administrator review and remediation.

org.openeai.CoreMessaging.Sync.Error-Sync

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

*.Create-Sync

A Create-Sync broadcasts the initial state of an object after it has been created in an authoritative application. The DataArea element of a Create-Sync contains a NewData element with the initial state of an object after it has been successfully created at the authoritative application, applying all of the application's business rules. As outlined in detail above, synchronization messages must be applied to non-authoritative applications in the proper chronological order and if a non-authoritative application encounters an error processing a Create-Sync, it must publish an org.openeai.CoreMessaging.Sync.Error-Sync message with appropriate errors. Errors can be of type "system" or "application", representing system level and application level errors. System level errors are more technical in nature and are usually sent with more technical descriptions.

Examples (follow links for message text)

Create-Syncs:

com.any-erp-vendor.Person.BasicPerson.Create-Sync
org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Create-Sync

*.Update-Sync

An Update-Sync broadcasts the changed state of an object after it has been updated in an authoritative application. The DataArea of an Update-Sync contains a BaselineData element with the state of an object prior to modification and a NewData element with the state of an object after it has been successfully modified at the authoritative application, applying all of the application's business rules.

When processing an Update-Sync message, a non-authoritative application may use the pre-modification state of the object in the BaselineData element to locate the appropriate object or records in the non-authoritative application to update to reflect the new state of the object contained in the NewData element of the message. Note that when processing an Update-Sync, a comparison of the current state of the object in the non-authoritative application with the state of the object in the BaselineData element of the message is technically not required in order to determine whether the Update-Sync message may be processed by the non-authoritative application. This is the case, because only non-authoritative applications should be consuming an Update-Sync, which means that the state of the object could not have been modified locally. However, although such a comparison is unnecessary if the integration has been implemented respecting the concept of authoritative source, such a comparison could be performed if desired, and appropriate error messages published as Error-Sync messages if there were a mismatch between the current state of the object in the non-authoritative application and the state of the object in the BaselineData element of the message. Additionally, the less-common process of merging the current version of the data in the non-authoritative system with the NewData provided by the authoritative source could be performed if desired.

As outlined in detail above, synchronization messages must be applied to non-authoritative applications in the proper chronological order, and if a non-authoritative application encounters an error processing an Update-Sync, it must publish an org.openeai.CoreMessaging.Sync.Error-Sync message with appropriate errors. Errors can be of type "system" or "application",

representing system level and application level errors. System level errors are more technical in nature and are usually sent with more technical descriptions.

Examples (follow links for message text)

Update-Syncs:

com.any-erp-vendor.Person.BasicPerson.Update-Sync
org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Update-Sync

*.Delete-Sync

A Delete-Sync instructs non-authoritative applications to delete an object locally that has been deleted from an authoritative application. The DataArea element of a Delete-Sync contains a DeleteData element that consists of a DeleteAction element and the current state of a message object to be deleted. The type attribute of the DeleteAction element specifies whether the requested delete is of type 'delete' or 'purge.' A DeleteAction of type 'delete' indicates that the application processing this request should attempt to perform a standard delete, applying all of its business rules. Many applications never actually remove objects or records from their stores. They inactivate them or perform some other appropriate business logic. A DeleteAction of type 'purge' is a special case, which directs the application processing the sync to actually purge or remove the object or records that comprise it from its store entirely. When implementing message support, you may chose to support or not support Delete-Syncs with a DeleteAction of type 'purge'. It may be totally inappropriate for some applications. However, it is very useful for automated setup and tear-down of test cases when testing messaging implementations with automated testing tools such as the OpenEAI TestSuiteApplication or other appropriate tools your organization may have. It is common to implement support for DeleteAction of type 'purge' and make that support configurable, so that it may be toggled on when used in development and test environments where testing may occur and toggled off in a production environment where sensitive records should never be completely purged from a system. If a DeleteAction of type 'purge' is not supported, the consuming application should publish an Error-Sync with an invalid delete action error (OpenEAI-1008) if a Delete-Sync with a DeleteAction of type 'purge' is received. If a DeleteAction of type 'purge' is supported, but toggled off, the consuming application should publish an Error-Sync with a purge disabled error (OpenEAI-1009).

When processing a Delete-Sync message, a non-authoritative application may use the state of the message object in the DeleteData element to locate the appropriate object or records in the non-authoritative application to delete. Note that when processing a Delete-Sync, a comparison of the current state of the object in the non-authoritative application with the state of the message object in the DeleteData element of the message is technically not required in order to determine whether the Delete-Sync message may be processed by the non-authoritative application. This is the case, because only non-authoritative applications should be consuming a Delete-Sync, which means that the state of the object could not have been modified locally. However, although such a comparison is unnecessary if the integration has been implemented respecting the concept of authoritative source, such a comparison could be performed if desired and appropriate errors published as Error-Sync messages if there were a mismatch between the current state of the object in the non-authoritative application and the state of the object in the DeleteData element of the message.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

As outlined in detail above, synchronization messages must be applied to non-authoritative applications in the proper chronological order, and if a non-authoritative application encounters an error processing a Delete-Sync, it must publish an org.openeai.CoreMessaging.Sync.Error-Sync message with appropriate errors. Errors can be of type “system” or “application”, representing system level and application level errors. System level errors are more technical in nature and are usually sent with more technical descriptions.

Examples (follow links for message text)

Delete-Syncs:

com.any-erp-vendor.Person.BasicPerson.Delete-Sync
org.any-openeai-enterprise.CoreApplication.InstitutionalIdentity.Delete-Sync

Common Error Messages

The following are error messages which arise from the OpenEAI Message Protocol and can be used when implementing message support. Many applications will also require their own special error messages to indicate error conditions that are specific to their purpose or business logic. The Reference Implementations department of the OpenEAI Project is beginning to maintain a list of all error messages which arise from the OpenEAI Message Protocol. In the future, the OpenEAI Project will post suggestions about which common error messages should be supported by every message application, gateway, and service that uses the OpenEAI Message Protocol and which are optional. The following are examples of error messages that have been used to date.

Code	Type	Description	Explanation
OpenEAI-1001	application	Unsupported message object: [unsupported message object name]. This application expects ‘[supported message object names]’.	Somehow the wrong message object name is getting placed into the message by the sending application, it is sending the wrong message, or the message is being sent to the wrong destination entirely.
OpenEAI-1002	application	Unsupported message action: [unsupported message action name]. This application only supports '[supported message action name(s)]' for message object ‘[relevant message object name]’.	Somehow the wrong message action name is getting placed into the message by the sending application, it is sending the wrong message, or the message is being sent to the wrong destination entirely.
OpenEAI-1003	application	Invalid query element found in the Query-Request message. This command expects '[name of the expected query object(s)]'.	A null or inappropriate query object is being placed into the message by the sending application or it is sending the wrong message entirely.
OpenEAI-1004	application	Inauthentic request message.	The request message does not appear to come from an application that is authorized to make the request.
OpenEAI-1005	application	Error authenticating request	Some type of error occurred while

		message. [provide error details].	attempting to authenticate the request message or sending application, so the message could not be proxied or processed.
OpenEAI-1006	application	Invalid identifier type found in the Identifier element of the Query-Request message. This application expects a type of '[Identifier type(s)]'.	When querying for certain types of objects, the query object used is a generic Identifier element with an appropriate type attribute. An incorrect identifier type appears in the message or the entirely wrong message was sent.
OpenEAI-1007	application	Null identifier value found in the Identifier element of the Query-Request message. This command expects that a value actually be present.	For some reason, no value is being placed in the Identifier element of the Query-Request message being sent.
OpenEAI-1008	application	Invalid delete action type. Only delete actions of type 'delete' and 'purge' are allowed.	Somehow an invalid delete action type is being set. Check the sending application or the appropriate Enterprise Objects document to figure out how this could be and prevent it from happening.
OpenEAI-1009	application	Purge disabled. The delete action type of purge is supported by this implementation, but it is presently disabled.	The application apparently supports a delete action of type 'purge', but this support is configurable, because there are conditions under which a purge should be allowed (such as test and development use) and should not be allowed (such as production use). Presently, this purge support is toggled off, presumably by an appropriate runtime configuration property.
OpenEAI-1010	application	Invalid delete element found in the Delete-Request message. This command expects an '[delete element name(s)]'.	Somehow there is an unsupported delete element in the message processed by this application. Verify that the sending application is sending the appropriate message and that it contains the appropriate delete element.
OpenEAI-1011	application	Invalid update element found in the NewData element of the Update-Request message. This command expects an '[update element name(s)]'.	Somehow there is an unsupported update element in the NewData element of the message processed by this command. Verify that the sending application is sending the appropriate message and that it contains the appropriate update elements in the NewData and Baseline data elements.
OpenEAI-1012	application	Invalid update element found in the BaselineData element of	Somehow there is an unsupported update element in the BaselineData

		the Update-Request message. This command expects an '[update element name(s)]'.	element of the message processed by this command. Verify that the sending application is sending the appropriate message and that it contains the appropriate update elements in the NewData and Baseline data elements.
OpenEAI-1013	application	Invalid create element found in the Create-Request message.	Somehow there is an unsupported create element in the message processed by this application. Verify that the sending application is sending the appropriate message and that it contains the appropriate create element.
OpenEAI-1014	application	Baseline is stale.	The baseline state of the message object in the message does not match the current state in the consuming application.

Enterprise Data Values

Now that you have taken an extensive look at the OpenEAI Message Protocol and the XML message format that it specifies, you may be wondering what data values actually go into the elements and attributes of these XML messages. In the answer to this question lies one of the most challenging and interesting aspects of practicing EAI—data value translations and data format transformations. The problem you probably face is that many applications in your enterprise and many of your trading partners have different valid values for data that is essentially supposed to be the same. For example, just think of how many different ways you have seen gender data represented over the years. F, M, N, U, male, female, neutered, unknown, not available, not reported, FE, ML, UK, NA, Male, Female, Unknown, 1, 2, 3, 5, 01, 02, 03. While we are sure that we have seen more valid values for gender in applications, the authors are definitely certain that we have seen all of the specific values listed here…and, no, we do not know why there were four possibilities for gender in a system, and the number 4 was apparently not an acceptable value, but 5 was!

Any EAI approach that you use must provide concepts, tools, and implementations to address this reality of divergent values and formats for essentially the same data. It would be helpful if you did not have to program these data translations and format transformations for each and every integration you build. Ideally, if you are taking the trouble to define your enterprise data objects, it would be great if you could define valid values to go with each and every field of those objects. Valid values are not really appropriate for some fields, formatting rules or masks are more appropriate. For example, your enterprise may have a preferred way to capitalize proper names using a simple or very complex set of rules, and you may have a preferred way to represent telephone numbers with hyphens and parentheses. Even though you may have preferred valid values and formatting rules, a number of your own legacy and purchased applications (as well as trading partners with whom you exchange data) have values and formatting rules that deviate from your preferred rules. So in addition to specifying your preferred valid values, formatting rules, and masks for each and every field of your enterprise data objects, you would like to specify valid values, rules, and masks for each of these fields in each and every

application with which you must integrate—and let some data translation and transformation foundation or infrastructure implement these configured translations and transformations for you. While we could dream about even more, this is a realistic nirvana: to be able to program application interfaces with the data values of the local application we are working on, knowing that the translations and transformations we configured or specified with our enterprise object definitions will simply be applied by foundation components or infrastructure applications. This is exactly what OpenEAI strives to provide with its enterprise object configuration documents and enterprise field foundation components.

The OpenEAI methodology recommends that you select and maintain a set of ***enterprise values*** for each field of every message object that you define. Keeping with the XML precepts of transparency and clarity, these enterprise values should be as obvious in their meaning as possible. Alternately, you might choose to take the values native to your most important enterprise applications—such as your ERP system—and dub them as your enterprise values. These will be the values that should appear in messages as they move about your enterprise, as these messages are serialized to enterprise messaging logs, and as these messages are used to present clear and usable information to end users in all of your new web applications without decoding or translating values. In other words, your enterprise values should be the clear and understandable values that human beings actually understand and use—values that have business meaning. Configurable software transforms alternate, archaic, or foreign values to your enterprise values as messages are built or as messages are applied to target applications.

The OpenEAI Methodology Document introduces the details of specifying enterprise values and translations using OpenEAI enterprise object configuration documents. The OpenEAI API Introduction Document presents the detailed structure of enterprise object configuration documents and the foundation components that use these documents to implement translations, transformations, scrubbing, and formatting.

Implementing Messaging Support

You probably already understand the interdependencies and general implications of implementing message support using the OpenEAI Message Protocol from reading the detailed behavioral descriptions for each message action and message type above. However, some of the implications may not be immediately clear, so this section serves as a summary and checklist for which message support to implement for authoritative and non-authoritative applications. For detailed information on how to implement message support according to the OpenEAI Message Protocol, you may move on to other documents in the OpenEAI Core Documentation Suite. Be sure to read the Getting Started with OpenEAI Document and then follow up with the OpenEAI API Introduction Document, the OpenEAI Methodology Document, and the OpenEAI Implementation Strategies Document in whichever order you think is helpful.

Message Support for Authoritative Applications

When an application is deemed authoritative for a message object, typically it must be prepared to handle and reply to the following messages of the OpenEAI Message Protocol for that object:

Query-Request
Create-Request and/or Generate-Request (if appropriate)

Update-Request
Delete-Request

The authoritative application must also be prepared to publish the following messages for that object:

Create-Sync
Update-Sync
Delete-Sync

Query-Request is implemented so that non-authoritative applications may query for the current state of an object. Create-Request is implemented so non-authoritative applications may create a new object in the authoritative application. Generate-Request is implemented if appropriate for the message object in question; that is, if the object in question can be generated by the authoritative application given some other object as input. Update-Request is implemented so non-authoritative applications may modify the state of objects at the authoritative source. Delete-Request is implemented so non-authoritative applications may delete objects at the authoritative source.

Some non-authoritative applications have no need to modify data objects of certain types, but they do need to know that these objects exist and may even need to maintain the states of these objects locally in order to operate. These are the applications that consume sync messages. For example, an ID card system may not allow a user to change his/her name, but this application need to know about this data and be updated with any changes in name that are made through the enterprise's ERP system.

Create-Sync is implemented so non-authoritative applications are informed of the creation of a new object and its initial state. Update-Sync is implemented so non-authoritative applications are informed of changes to an existing object's state. Delete-Sync is implemented to instruct non-authoritative applications to delete objects that have been deleted at the authoritative application. These sync messages must be published by an authoritative application regardless of how an object is created, updated, or deleted in the authoritative application, through messaging or through any other means of creating, modifying, or deleting data in the authoritative application.

Any given application can definitely be authoritative for some message objects and not authoritative for others. Clearly, more or less of this message support can be implemented based on the nature of the application and the needs of the enterprise. However, it is important to note that the OpenEAI methodology recommends fully implementing the OpenEAI Message Protocol for each message object whenever appropriate and feasible. Complete implementation provides the most flexibility for the future. This practice positions an enterprise to perform more complex integrations more quickly later. After all, that flexibility is largely what a solid EAI strategy is all about—increasing the agility of organizations, allowing them to do more and do it more quickly.

Message Support for a Non-Authoritative Application

Depending on the needs of the non-authoritative application, it may be required to send the following requests and handle the replies that are returned:

Query-Request
Create-Request and/or Generate-Request (if appropriate)
Update-Request

Delete-Request

Sending these requests and handling their replies is required only if the non-authoritative application needs to modify the state of objects for which another application is authoritative. If the non-authoritative application does not have a need to modify the state of objects for which another application is authoritative, but rather it just needs to know the state of these objects at their authoritative source, then it only needs to be prepared to consumer and process the following sync messages:

Create-Sync
Update-Sync
Delete-Sync

Again, more or less of this message support can be implemented based on the nature of the application and the needs of the enterprise. However, implementation of support for all actions is recommended to afford the highest level of flexibility.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

Appendix 1: The GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly

within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further

copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy

of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice.

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in

the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

$Revision: 1.28$
$Date: 3/11/2003 2:07:40 PM$
$Source: /cvs/repositories/openeai/project/documentation/core/MessageProtocol.doc$

The Free Software Foundation may publish new, revised versions
of the GNU Free Documentation License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns. See
http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number.
If the Document specifies that a particular numbered version of this
License "or any later version" applies to it, you have the option of
following the terms and conditions either of that specified version or
of any later version that has been published (not as a draft) by the
Free Software Foundation. If the Document does not specify a version
number of this License, you may choose any version ever published (not
as a draft) by the Free Software Foundation.