XML ARCHITECTURE

2.0 Introduction

The World Wide Web Consortium (W3C) has released the first version of the Extensible Markup Language (XML) specification to broad support within the industry. XML was designed to fill a deficiency identified in other Internet and markup standards, like HTML, by supporting data definition and information processing requirements. It provides a middle ground between the restrictions and general inflexibility of HTML, and the overall complexity of SGML. This specification has become widely supported as the beginning of the next generation of Internet capabilities. Extensions and enhancements to the XML specification are under development within the W3C. As these are completed and released, they will be reviewed by the Open Applications Group for applicability in support of further enabling application to application integration.

The Open Applications Group has developed an approach for implementation of the Open Applications Group Interface Specification (OAGIS) within an XML framework. This section describes the process for development, translation, and implementation of OAGIS compliant Business Object Documents in XML. For detailed information on the XML specification itself, and other related W3C projects, please refer to the official W3C web site at www.w3c.com, or one of the many available publications on the topic.

At this time, implementation projects using the OAGIS may be based on the original message structure, or in the XML format described in this document. The decision to use one format or another is left to the systems integrators, application vendors, and ultimately, the customers. It is expected that the need for support of both approaches will decrease over time, as migration to XML occurs, and software vendors begin to support XML as a native integration mechanism within their

products. As migration and adoption progresses, the proprietary format may be phased out.

The Open Applications Group is aware of the broad set of initiatives around XML technology. While the results of many of these initiatives are directly applicable to support of the integration specification, only released recommendations will be used to implement the content of the specification. We hope this will allow the reference files distributed to our members and supporters to remain as stable as possible. As enhancement opportunities are identified, and implementation bugs are reported, each of the reference files will be updated as appropriate. Additionally, other tools and resources may be provided to support XML implementations of the specification.

As an overall design principle, the Open Applications Group's approach to XML is focused on simplicity and usability. We have tried to provide the entire body of knowledge contained in the specification within the set of distributed DTDs, while avoiding the use of some of the more sophisticated XML constructs unless absolutely necessary.

While this document reviews each major component of the Interface Specification as implemented in XML, it is not intended to be a primer or training guide. A strong working knowledge of both XML and the OAGIS is assumed.

In the sections that follow, the case for XML is outlined, followed by the steps required to incorporate support for XML within the specification. Sample files are provided as a part of the discussion. Additionally, guidelines for extending the specification using the USERAREA are provided. Unique transaction pairs, such as Get, Show, GetList, and List are discussed as well. The entire set of XML DTDs are available from the Internet web site at www.openapplications.org.

For additional information, to report bugs, or request enhancements, please send e-mail to xml@openapplications.org.

2.1 The Case for XML

XML provides significant advantages over more traditional approaches to inter-application messaging and interface development. These advantages make a compelling case for its use in support of the OAGIS. The result is a framework that strongly supports the implementation needs of the vendor, integrator, and customer communities. A number of considerations in this area are provided here.

3rd Party Tool and Development Support: Vendors and customers may leverage the rapidly expanding market of available 3rd party tools and development libraries as a part of their implementation solutions. These include parsing and validating APIs and message generation tools. As the XML standard continues to be expanded, solution providers will be well positioned to take advantage of the applicable enhancements with little incremental internal development effort.

Readability: Traditional messaging and interface techniques have often been implemented using cryptic message formats. Development and troubleshooting these solutions is correspondingly difficult. XML is easy to read and structured intuitively.

Extensibility: A unique characteristic of an XML implementation is its support for vendor and implementation specific extensions. While the OAGIS is robust, there will be a number of circumstances where implementations need to support data elements that are not a part of the specification. These extensions may be incorporated into the solution without adding complexity or increasing maintenance effort. Additionally, extensions from multiple parties may be incorporated concurrently. Details are provided later in this document.

Knowledge and Skills Base: The widespread awareness and adoption of XML has resulted in a significant knowledge base for customer and vendor support. Unlike proprietary solutions, knowledge and support of integration solutions based on XML may be obtained throughout the industry. As XML gains in popularity, the available resource pool promises to continue to grow.

Related Technologies: Another advantage of an XML implementation is the opportunity for incorporation into other types of solutions based on the same technology. These include web browser support, programming tools, and knowledge management systems. A number of initiatives in these areas are already underway.

In general, the Open Applications Group's use of XML will be limited to the standard implementation as released by the World Wide Web Consortium. In order to maintain flexibility and openness, the use of proprietary extensions or non-standard constructs will not be supported.

2.2 Implementation Overview

Within the scope of business application integration, XML solutions are comprised of two key components; the DTDs (Document Type Definitions), and the XML transaction messages themselves. *(XSL style sheets are not used for the current implementation.)* The DTD is used to formally define and validate the overall structure and content of a message. Simply put, the DTD acts as a template used to define the message structure and relationships between the data elements. In some cases, the DTD information is embedded inside the XML message. For the purposes of OAGIS XML implementation, DTDs will always be deployed as a set of separate files.

According to the W3C Frequently Asked Questions list,

> *"A DTD is usually a file (or several files to be used together) which contains a formal definition of a particular type of document. This sets out what names can be used for elements, where they may occur, and how they all fit together."*

Mapping from the OAGIS specification to XML requires the development of a set of DTDs. The sections below describe the process for developing DTDs to support OAGIS messages, the structure of the DTDs that have been developed and will be released to validate messages, and the conversion of a few sample messages into XML.

The OAGIS XML solution is implemented as a set of three resource DTDs, and an additional DTD for each service request. The resource DTDs include information that is common across all requests. Resource DTDs are used to define Data Types, Fields, and Segments. A small number of complex data types, or Super Segments are also defined. The service request support files define their unique definition and structure, as well as additional attributes or element restrictions that are not defined in the resource files. Each of these topics is discussed in more detail in later sections of this document.

2.3 Resource DTD – Data Domains

The first resource file used in XML mapping is the data type definition file, **oagis_domains.dtd**. This file describes the data types used in all subsequent DTDs.

Below is the entire DTD for data type domains. We'll discuss each component of the resource file below. Most of this file is comments and revision control information. The actual element specification for the string domain is a single line.

```
<!--
    $Revision: 6.2.0 $
    $Date: 12 December 1998 $
    Open Applications Group DTD
    Copyright 1998, All Rights Reserved

    $Name: oagis_domains.dtd $
-->

<!-- ================================================== -->
<!-- Entities - Domains
    This section defines data domains for the
    primitive element values.  This is the only
    area where the primitive #PCDATA should
    appear.
-->

<!-- String Data: Generic Character Data Domain -->
<!ENTITY % STRDOM "(#PCDATA)">
```

Before working through the OAGIS-specific details, a few notes on XML in general must be provided.

Delimiters: Components of a DTD are defined by a set of delimiters. In the example above, two are introduced: comments and entities.

Comments: Comments within a DTD are delimited by the tokens **<!--** and **-->**. Information between these tokens is considered a comment and is ignored. Comments may cross multiple lines.

Entities: Entities within a DTD are delimited by the tokens **<!ENTITY** and **-->**. In many cases, entities are used to define replacement text within the DTDs.

The STRDOM Domain

The reference DTD provided above is mostly comments, including revision information and documentation. A single entity called **STRDOM** is defined. This

entity is defined as **#PCDATA**, or parsed character data. Simply put, each field can be composed of any character information supported by XML.

Entity definitions act similarly to traditional macros for replacement text. References to these entities appear later in the specification DTDs. In the string domain specification below, instances of the **"%STRDOM;"** entity are replaced with the text **(#PCDATA)** as the XML file is interpreted.

```
<!ENTITY % STRDOM "(#PCDATA)">
```

At this time, only string data domains are implemented in the specification. Over time, additional typing may be introduced to further support validation. Representative data type domains that may be introduced later include simple definitions such as dates or numbers (integer and/or float). More complex domains that may be appropriate may include phone numbers, or even sets of ISO codes.

It is expected that the evolution of XML, and facilities for stronger data typing as has been proposed through efforts such as XML-Data, will begin to address these topics directly. The Open Applications Group implementation will continue to review, and adopt as appropriate, alternative approaches as they are released as industry standards.

2.4 Resource DTD – Fields

The next resource file used in XML mapping provides field element definitions. These are the next most primitive element used in OAGIS implementation in XML. Field definitions are shared across all segment and transaction DTDs.

Below is a small portion of the field DTD, **oagis_fields.dtd**. In the distributed DTD, all specification fields are included.

```
<!--
    $Revision: 6.2.0 $
    $Date: 12 December 1998 $
    Open Applications Group DTD
    Copyright 1998, All Rights Reserved

    $Name: oagis_fields.dtd $
-->

<!-- ================================================== -->
<!-- Entities - Fields - Appendix C
-->

<!ELEMENT ACCTPERIOD %STRDOM;>
<!ELEMENT ACCTYEAR %STRDOM;>
<!ELEMENT ACKREQUEST %STRDOM;>
<!ELEMENT ADDRLINE %STRDOM;>
<!ELEMENT ADDRTYPE %STRDOM;>
<!ELEMENT AUTHID %STRDOM;>
<!ELEMENT BUSNAREA %STRDOM;>
<!ELEMENT CHARGEID %STRDOM;>
<!ELEMENT CONFIRMATION %STRDOM;>
      ...
```

The element definitions provided above define the specification fields. Each field is defined as type **STRDOM**. The **STRDOM** entity is replaced with the definition provided in the domain reference file described above. So, the **ACCTPERIOD** element

```
<!ELEMENT ACCTPERIOD %STRDOM;>
```

is parsed with the domain definition defined above, and is interpreted as the following:

```
<!ELEMENT ACCTPERIOD (#PCDATA)>
```

The one exception the domain definitions is the content model of the USERAREA element. The content model for this element is ANY, as shown below. This allows for customer and vendor specific extensions to be incorporated into the DTDs as described in the implementation notes later in this document.

```
<!ELEMENT USERAREA ANY>
```

2.5 Resource DTD – Segments

The third resource file, **oagis_segments.dtd**, defines OAGIS segments and super segments that are shared across all transactions. Segments are complex data types that contain more than one element. Super Segments are simply segments that may contain multiple level content definitions.

One representative segment, AMOUNT, is shown below. The example is followed by a brief description of each key component.

```
<!--
    $Revision: 6.2.0 $
    $Date: 12 December 1998 $
    Open Applications Group DTD
    Copyright 1998, All Rights Reserved

    $Name: oagis_segments.dtd $
-->

<!-- ================================================== -->
<!-- Elements - Segments - Appendix D
-->

<!-- AMOUNT -->
<!ENTITY % SEG_AMOUNT_QUALIFIER_EXTENSION "OTHER">
<!ENTITY % SEG_AMOUNT_QUALIFIERS
    "(ACTUAL | APPRVORD | AVAILABLE | BUDGET | DISCNT1 |
      DISCNT2 | DISCNT3 | DISCNT4 | DISCNT5 | DISCNT6 | DISCNT7 |
      DISCNT8 | DISCNT9 | DOCUMENT | EXTENDED | ITEM | OPENITEM |
      ORDER | ORDLIMIT | TAX | TAXBASE | TOTLIMIT |
       %SEG_AMOUNT_QUALIFIER_EXTENSION;)">
<!ENTITY % SEG_AMOUNT_TYPES_EXTENSION "OTHER">
<!ENTITY % SEG_AMOUNT_TYPES
    "(T | F | %SEG_AMOUNT_TYPES_EXTENSION;)">
<!ELEMENT AMOUNT (VALUE, NUMOFDEC, SIGN, CURRENCY, DRCR)>
<!ATTLIST AMOUNT
    qualifier %SEG_AMOUNT_QUALIFIERS; #REQUIRED
    type %SEG_AMOUNT_TYPES; #REQUIRED
    index CDATA #IMPLIED>
```

Each segment has a set of supporting attributes. All segments contain the **QUALIFIER** attribute, and many have **TYPE** and **INDEX** attributes as well. The **QUALIFIER** is defined as an entity below. In the definition of the **AMOUNT** element, this definition is referenced. The entity definition lists the set of valid values for the attribute.

```
<!ENTITY % SEG_AMOUNT_QUALIFIERS
    "(ACTUAL | APPRVORD | AVAILABLE | BUDGET | DISCNT1 |
      DISCNT2 | DISCNT3 | DISCNT4 | DISCNT5 | DISCNT6 | DISCNT7 |
      DISCNT8 | DISCNT9 | DOCUMENT | EXTENDED | ITEM | OPENITEM |
      ORDER | ORDLIMIT | TAX | TAXBASE | TOTLIMIT |
       %SEG_AMOUNT_QUALIFIERS_EXTENSION;)">
```

The second entity that is used to support the **AMOUNT** element is the set of values used in attribute **TYPE**. These values (transactional and functional) are defined below.

```
<!ENTITY % SEG_AMOUNT_TYPES
    "(T | F | %SEG_AMOUNT_TYPES_EXTENSION;)">
```

The **AMOUNT** element is comprised of six (6) required fields. The definitions of these fields were provided in the resource file **oagis_fields.dtd** above. None of these fields are optional, or occur more than once.

```
<!ELEMENT AMOUNT (VALUE, NUMOFDEC, SIGN, CURRENCY, DRCR)>
```

The **AMOUNT** element also has an associated set of attributes. They are both required in the XML message. The allowed values are defined through the use of entities **SEG_AMOUNT_QUALIFIERS** and **SEG_AMOUNT_TYPES**.

```
<!ATTLIST AMOUNT
    qualifier %SEG_AMOUNT_QUALIFIERS; #REQUIRED
    type %SEG_AMOUNT_TYPES; #REQUIRED
    index CDATA #IMPLIED>
```

The remaining segments are defined in the resource DTD with a similar structure. Most of them contain the **QUALIFIER** and **TYPE** attributes.

SEGMENT ENTITIES

As may be apparent from the list of qualifiers in the segment sample shown above, a segment may be used multiple times within the same data type, each with a different qualifier or qualifier/type combination.

One example of these entity definitions is shown below. One of the qualifiers for the **AMOUNT** segment is **ACTUAL**. Additionally, this qualifier is implemented with both transactional and functional types (**T** and **F**).

```
<!ENTITY % AMOUNT.ACTUAL.F    "AMOUNT">
<!ENTITY % AMOUNT.ACTUAL.T    "AMOUNT">
```

The current specification includes 29 unique qualifier and type combinations for the **AMOUNT** segment. The XML implementation of the OAGIS represents a trade-off between creating a large number of elements in the specification and not clearly defining which qualifiers are to be used in each BOD. This release represents a middle ground between the two approaches.

The result is that each transaction DTD provides a definition of which qualifier and/or type is used in each location. This definition is provided in the form of parameter entities as shown above. The transaction DTD fragment shown below demonstrates how the entities are implemented.

A business analyst or programmer may then use this information to support design and development activities around each transaction. However, when the final XML file is actually transmitted, the entity definition is interpreted simply as the core segment. In this case, the **AMOUNT** element is processed in both cases.

```
<!ELEMENT JELINE (
  (%AMOUNT.ACTUAL.T;),
  (%AMOUNT.ACTUAL.F;)?,
  GLNOMACCT, BUSNAREA?, COSTCENTER?, DEPARTMENT?, DESCRIPTN?, …
```

"SUPER SEGMENTS"

In a small number of instances, more complex data types should be consistently defined across the entire specification. Typically, these will represent very general information structures, such as partner addresses, charges, and charge distributions. These may be defined with multiple levels, possibly incorporating multiple segments or data types. These elements are generally referred to as Super Segments.

One example of a Super Segment is provided below. The **CHARGE** segment includes another segment (**OPERAMT**), a number of fields, and another Super Segment (**DISTRIBUTN**).

```
<!ELEMENT CHARGE (
  (%OPERAMT.EXTENDED.T;)?,
  CHARGEID?, CHGLINENUM?, DESCRIPTN?, USERAREA?, DISTRIBUTN*)>
```

2.6 Sample Transaction DTD - Confirm BOD

The three resource DTDs contain common definitions that are shared across all of the transactions within the specification. Definitions that are unique to individual transactions are provided in the corresponding transaction DTD. A separate DTD file is distributed for each transaction. One of the simplest transactions in the specification, version 2 of the Confirm BOD, is described in this section. The complete DTD, **`002_confirm_bod_002.dtd`**, is shown below.

The naming convention used for transaction DTD files is "`<Chapter>_<Verb>_<Noun>_<Version>.dtd`".

```
<!--
    $Revision: 6.0.0 $
    $Date: 15 June 1998 $
    Open Applications Group DTD
    Copyright 1998, All Rights Reserved

    $Name: 002_confirm_bod_002.dtd $
-->

<!-- ================================================== -->
<!ENTITY % DOMAINS SYSTEM "oagis_domains.dtd">
%DOMAINS;

<!ENTITY % FIELDS SYSTEM "oagis_fields.dtd">
%FIELDS;

<!ENTITY % SEGMENTS SYSTEM "oagis_segments.dtd">
%SEGMENTS;

<!-- ================================================== -->

<!ELEMENT BOD (CNTROLAREA, DOCUMENT+)>

    <!ATTLIST VERB value CDATA #FIXED "CONFIRM">
    <!ATTLIST NOUN value CDATA #FIXED "BOD">
    <!ATTLIST REVISION value CDATA #FIXED "002">

<!ELEMENT DOCUMENT (CONFIRM_BOD)>

    <!ELEMENT CONFIRM_BOD (CONFIRM, CONFIRMMSG*)>

        <!ELEMENT CONFIRM (SENDER, STATUSLVL, DESCRIPTN?,
          ORIGREF?, USERAREA?)>

        <!ELEMENT CONFIRMMSG (DESCRIPTN?, REASONCODE?,
          USERAREA?)>
```

In the first section of the transaction DTD, resource information is loaded from the data domains, fields, and segments reference files. In order to load this information, entities are created to reference the external definitions. When the DTD is parsed, the contents of the external files replace the entity references.

```
<!ENTITY % DOMAINS SYSTEM "oagis_domains.dtd">
%DOMAINS;

<!ENTITY % FIELDS SYSTEM "oagis_fields.dtd">
%FIELDS;

<!ENTITY % SEGMENTS SYSTEM "oagis_segments.dtd">
%SEGMENTS;
```

Next, the root, or top-level, message element is defined. The same definition is used for all transactions. The **BOD** element contains two sub-elements. The **CNTROLAREA** sub-element occurs exactly once. The **DOCUMENT** sub-element occurs one or many times. The **CNTROLAREA** element is an OAGIS segment, and is defined in the segment resource file.

```
<!ELEMENT BOD (CNTROLAREA, DOCUMENT+)>
```

Additional attributes for **CNTROLAREA** elements are defined to restrict the values that may be provided in the **VERB**, **NOUN**, and **REVISION** elements. When the XML message is generated, the contents of these elements must exactly match the **#FIXED** values listed.

```
<!ATTLIST VERB value CDATA #FIXED "CONFIRM">
<!ATTLIST NOUN value CDATA #FIXED "BOD">
<!ATTLIST REVISION value CDATA #FIXED "002">
```

After the **CNTROLAREA** is defined, the rest of the DTD describes the structure of the transaction data. The **DOCUMENT** element is always defined as the verb-noun combination of the transaction. This verb-noun element, **CONFIRM_BOD** in this example, is then developed from its member data types defined in the OAGIS.

The first OAGIS data type is represented by the **CONFIRM** element. This element contains two required elements and three optional elements. These elements are defined in the field resource file.

```
<!ELEMENT DOCUMENT (CONFIRM_BOD)>

    <!ELEMENT CONFIRM_BOD (CONFIRM)>

        <!ELEMENT CONFIRM (SENDER, STATUSLVL, DESCRIPTN?,
          ORIGREF?, USERAREA?)>
```

The remainder of the transactions defined in the OAGIS follow the same structure as the CONFIRM BOD just described. These transactions are more complex. In

many cases, they require deeper element nesting and more sophisticated regular expressions.

2.7 Sample Message – Confirm BOD

XML messages are constructed based upon the definitions provided in the DTDs. When the message is parsed, it is compared against the DTDs to validate that it is valid and well formed. Each required element must be present in the message, and the elements must be sent in the correct order.

An example of the CONFIRM BOD message implemented in XML follows. The four main components of this message are described below.

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

<!DOCTYPE confirm_bod_002 SYSTEM "002_confirm_bod_002.dtd">

<BOD>
    <CNTROLAREA>
        <BSR>
            <VERB>CONFIRM</VERB>
            <NOUN>BOD</NOUN>
            <REVISION>002</REVISION>
        </BSR>
        <SENDER>
            <LOGICALID>XXX1234YYY</LOGICALID>
            <COMPONENT>G/L</COMPONENT>
            <TASK>CONFIRM</TASK>
            <REFERENCEID>REF1</REFERENCEID>
            <CONFIRMATION>0</CONFIRMATION>
            <LANGUAGE>EN</LANGUAGE>
            <CODEPAGE/>
            <AUTHID>JOE DOE</AUTHID>
        </SENDER>
        <DATETIME qualifier = "CREATION">
            <YEAR>1995</YEAR>
            <MONTH>12</MONTH>
            <DAY>31</DAY>
            <HOUR>17</HOUR>
            <MINUTE>59</MINUTE>
            <SECOND>00</SECOND>
            <SUBSECOND>0000</SUBSECOND>
            <TIMEZONE>-0500</TIMEZONE>
        </DATETIME>
    </CNTROLAREA>
    <DOCUMENT>
        <CONFIRM_BOD>
            <CONFIRM>
                <SENDER>
                    <LOGICALID>XX141HG09</LOGICALID>
                    <COMPONENT>INVENTORY</COMPONENT>
                    <TASK>RECEIPT</TASK>
                    <REFERENCEID>95129945823449</REFERENCEID>
                    <CONFIRMATION/>
                    <LANGUAGE>EN</LANGUAGE>
                    <CODEPAGE></CODEPAGE>
                    <AUTHID>JOE DOE</AUTHID>
                </SENDER>
                <STATUSLVL>00</STATUSLVL>
                <DESCRIPTN>PROCESSED WITHOUT ERRORS</DESCRIPTN>
```

```
                <ORIGREF>RCPT#12550699</ORIGREF>
            </CONFIRM>
        </CONFIRM_BOD>
    </DOCUMENT>
</BOD>
```

The first line of the message includes the XML release version that was used. For this example, version 1.0 is indicated. The processor will parse the message based upon the 1.0 specification as released by the W3C and implemented in the XML tools.

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

The next file segment defines the document type (**DOCTYPE**) element. This element is defined with three components. The first component is the name of the document type, in this example, "**002_confirm_bod_002**". The second component is the access path for the definition file, in this example, "**SYSTEM**". This indicates that the DTD location is an operating system path. XML also supports references to URLs instead of local operating system file locations. The third and final component is the file name of the DTD, "**002_confirm_bod_002.dtd**".

```
<!DOCTYPE confirm_bod_002 SYSTEM "002_confirm_bod_002.dtd">
```

The next set of lines contain control information for the message. All transactions include this information, in the same format. The **BSR** element contains the verb, noun, and version definitions that match the transaction type. **SENDER** and **DATETIME** elements are provided as well.

The actual transaction data is included between the **DOCUMENT** tags. These elements are discussed below.

```
<BOD>
    <CNTROLAREA>
        <BSR>
            <VERB>CONFIRM</VERB>
            <NOUN>BOD</NOUN>
            <REVISION>002</REVISION>
        </BSR>
        <SENDER>
            <LOGICALID>XXX1234YYY</LOGICALID>
            <COMPONENT>G/L</COMPONENT>
            <TASK>CONFIRM</TASK>
            <REFERENCEID>REF1</REFERENCEID>
            <CONFIRMATION>0</CONFIRMATION>
            <LANGUAGE>EN</LANGUAGE>
            <CODEPAGE></CODEPAGE>
            <AUTHID>JOE DOE</AUTHID>
        </SENDER>
```

```
        <DATETIME qualifier = "CREATION">
            <YEAR>1995</YEAR>
            <MONTH>12</MONTH>
            <DAY>31</DAY>
            <HOUR>17</HOUR>
            <MINUTE>59</MINUTE>
            <SECOND>00</SECOND>
            <SUBSECOND>0000</SUBSECOND>
            <TIMEZONE>-0500</TIMEZONE>
        </DATETIME>
    </CNTROLAREA>
    <DOCUMENT>

    </DOCUMENT>
</BOD>
```

Transaction data is provided in the **DOCUMENT** definition. Each element provided is contained within a pair of tags, as defined in the DTD.

```
        <CONFIRM_BOD>
            <CONFIRM>
                <SENDER>
                    <LOGICALID>XX141HG09</LOGICALID>
                    <COMPONENT>INVENTORY</COMPONENT>
                    <TASK>RECEIPT</TASK>
                    <REFERENCEID>95129945823449</REFERENCEID>
                    <CONFIRMATION>0</CONFIRMATION>
                    <LANGUAGE>EN</LANGUAGE>
                    <CODEPAGE></CODEPAGE>
                    <AUTHID>JOE DOE</AUTHID>
                </SENDER>
                <STATUSLVL>00</STATUSLVL>
                <DESCRIPTN>PROCESSED WITHOUT ERRORS</DESCRIPTN>
                <ORIGREF>RCPT#12550699</ORIGREF>
            </CONFIRM>
        </CONFIRM_BOD>
```

Each XML formatted message will have the same structure. The main components are the version header, DTD reference, control information, and transaction data. Each of these was briefly discussed above.

2.8 DTD Development Steps – Resource Files

Development of the XML resources required for OAGIS support is most easily facilitated by review and conversion of the individual transaction files. This section describes the process to code the transactions in XML.

As a part of the conversion, some information that is provided in the specification is not directly supported by XML. These items will be noted in the discussion. The conversion process and resource file format will be enhanced over time to incorporate as much of this information as possible. These enhancements will be developed through adoption of additional capabilities as they are released by the W3C.

Domain Resource DTD Definitions

The domain resource file contains data type and value restrictions. In the current XML implementation, only one domain is defined.

Each field that is defined in the specification is a member of a data domain. The XML implementation of this domain is shown below. This string domain is currently used for all fields. Later, the DTDs may be extended to support the additional attributes defined in the OAGIS.

```
<!ENTITY % STRDOM "(#PCDATA)">
```

All fields in the specification DTDs reference the **STRDOM** definition.

Field Resource DTD Definitions

The next resource area provided is the set of field definitions. When implemented in XML, the name and primitive data type are provided.

```
<!ELEMENT DRCR %STRDOM;>
```

Each field that is used in the specification is provided in the field resource DTD. Fields used in segments are handled in the same manner as those defined directly in the transaction data types. Field alignment, padding, and length restrictions are described in the specification, but are not used in the XML translation.

Segment Resource DTD Definitions

The third resource that is provided in the XML implementation is the set of segment definitions. These definitions provide the content model for the segment, as well as the qualifiers and types that may be used in conjunction with the segment.

```
<!ENTITY % SEG_AMOUNT_QUALIFIERS
    "(ACTUAL | APPRVORD | AVAILABLE | BUDGET | DISCNT1 |
      DISCNT2 | DISCNT3 | DISCNT4 | DISCNT5 | DISCNT6 | DISCNT7 |
      DISCNT8 | DISCNT9 | DOCUMENT | EXTENDED | ITEM | OPENITEM |
      ORDER | ORDLIMIT | TAX | TAXBASE | TOTLIMIT |
       %SEG_AMOUNT_QUALIFIERS_EXTENSION;)">
<!ENTITY % SEG_AMOUNT_TYPES
    "(T | F | SEG_AMOUNT_TYPES_EXTENSION;)">
<!ELEMENT AMOUNT (VALUE, NUMOFDEC, SIGN, CURRENCY, DRCR)>
<!ATTLIST AMOUNT
    qualifier %SEG_AMOUNT_QUALIFIERS; #REQUIRED
    type %SEG_AMOUNT_TYPES; #REQUIRED
    index CDATA #IMPLIED>
```

Each segment defined in the OAGIS is provided in the segment resource file, **`oagis_segments.dtd`**. These are then incorporated into each of the transaction DTDs.

2.9 DTD Development Steps – Transactions

The differences in the way transaction files are defined in the traditional Open Applications Group format and XML are significant. XML allows for a very flexible definition of the overall structure, with straightforward support for optional and multiple fields embedded in the message data. The original structure of an OAGIS format message was to provide sets of header meta-data to describe the message fields.

Control Area DTD Definitions

The first section of the DTD provides control information, while the second includes the actual transaction data. The **CNTROLAREA** is defined as a segment. The **DATAAREA** element is discussed in the next section.

The BOD element in the XML example below is defined as required and single instance. The very simple regular expression, "**CNTROLAREA**" indicates that it is required and only occurs once. In the same fashion, the required and possibly multi-occurrence transaction data area is represented by the simple regular expression "**DATAAREA+**".

```
<!ELEMENT BOD (CNTROLAREA, DATAAREA+)>

    <!ATTLIST VERB value CDATA #FIXED "CONFIRM">
    <!ATTLIST NOUN value CDATA #FIXED "BOD">
    <!ATTLIST REVISION value CDATA #FIXED "001">

       <!ELEMENT DOCUMENT (CONFIRM_BOD)>
```

Additional attributes are defined in the header information. The **VERB**, **NOUN**, and **REVISION** attributes are required for all transactions, and must match the corresponding, appropriate transaction values. The remainder of the **CNTROLAREA** element definition is provided in the segment resource file, **oagis_segments.dtd**. The structure of the transaction information is discussed in the next section.

Transaction DTD Definitions

The **DOCUMENT** element contains a reference to only a single element created from the transaction's verb and noun. In this case, the element is **CONFIRM_BOD**. The **CONFIRM_BOD** element is then defined with its member data types.

Segments and fields are defined in the reference files, so the element definition simply lists the order of the fields, which are required, and which may have multiple instances. In the example below, two (2) fields and required, and three (3) are optional. Each may occur only once.

```
<!ELEMENT BOD (CNTROLAREA, DOCUMENT+)>

    <!ATTLIST VERB value CDATA #FIXED "CONFIRM">
    <!ATTLIST NOUN value CDATA #FIXED "BOD">
    <!ATTLIST REVISION value CDATA #FIXED "001">

       <!ELEMENT DOCUMENT (CONFIRM_BOD)>

           <!ELEMENT CONFIRM_BOD (CONFIRM)>

              <!ELEMENT CONFIRM (SENDER, STATUSLVL, DESCRIPTN?,
                ORIGREF?, USERAREA?)>
```

XML DTD Development Steps

Generating DTDs from the OAGIS is a straightforward process. The steps in the conversion are described below. Many of the constructs have already been described.

1. Update **CNTROLAREA** attributes for the transaction. The **VERB**, **NOUN**, and **REVISION** values must match the transaction.

```
<!ATTLIST VERB value CDATA #FIXED "VERB">
<!ATTLIST NOUN value CDATA #FIXED "NOUN">
<!ATTLIST REVISION value CDATA #FIXED "REVISION">
```

2. Create transaction element within **DOCUMENT**. The **DOCUMENT** entity definition includes only one element. This element is the concatenation of the transaction verb and noun, with an underscore between.

```
<!ELEMENT DOCUMENT (VERB_NOUN)>
```

3. Add segments for Data Types.

3.1 Add required segments in the element definition.

```
<!ELEMENT SAMPLE (REQSEGMENT)>
```

3.2 Add optional segments in the element definition.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?)>
```

4. Add fields in the element definition.

4.1 Add required fields in the element definition.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?,
REQFIELD1, REQFIELD2+)>
```

4.2 Add optional fields in the element definition.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?,
REQFIELD1, REQFIELD2+, OPTFIELD1?,
OPTFIELD2*)>
```

5. Add **USERAREA** element.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?,
REQFIELD1, REQFIELD2+, OPTFIELD1?,
OPTFIELD2*, USERAREA?)>
```

6. Add Rules. If processing rules are provided for the Data Type, the regular expressions describing these rules are coded next. These are added to the DTD immediately before the USERAREA element.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?,
REQFIELD1, REQFIELD2+, OPTFIELD1?,
OPTFIELD2*, (OPTFIELD1 | OPTFIELD2),
USERAREA?)>
```

In some cases, processing notes may indicate that a combination of fields are allowed, but only one of the fields listed in the set is required in a valid transaction. A more complex rule is provided to represent this processing note.

For example, the specification may indicate that at least one of the following fields is required: {**DESCRIPTN**, **ITEMID**, **ITEMIDX**, **SKU**}. The corresponding rule as coded in the transaction DTD as follows:

```
((DESCRIPTN, ITEMID, ITEMIDX?, SKU?) |
 (ITEMID, ITEMIDX?, SKU?) |
 (ITEMIDX, SKU?) |
 (SKU))
```

This content model is structured to ensure that at least one of the required fields is provided in the XML message, and the message contents satisfy exactly one of the model alternatives.

6.1 Remove elements included in rules. Elements that are parts of a rule definition are removed from the parent element.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?,
REQFIELD1, REQFIELD2+, (OPTFIELD1 |
OPTFIELD2*), USERAREA?) >
```

7. Add Sub-Data Type elements. The last elements that are added to the definition are references to sub-elements.

```
<!ELEMENT SAMPLE (REQSEGMENT, OPTSEGMENT?,
REQFIELD1, REQFIELD2+, (OPTFIELD1 |
OPTFIELD2), USERAREA?, SAMPLE_CHILD1,
SAMPLE_CHILD2)>
<!ELEMENT SAMPLE_CHILD1 (…)>
<!ELEMENT SAMPLE_CHILD2 (…)>
```

8. Repeat steps 3 through 8 for each element.

2.10 Get Transactions

The Get series of transactions is handled differently from the other transactions. It is used to request specific information from another system. As a result, each field and data type is optional. Those fields that are transmitted as a part of the Get BOD are used for selection on the receiving system.

The fields used for selection, and those requested to be returned may be different. In order to support this functionality, an additional attribute, **returndata**, is defined.

```
<!ATTLIST FIELDNAME returndata CDATA #REQUIRED>
```

Each field in a Get transaction requires the **returndata** attribute. This attribute indicates whether the data type should be populated by the receiving application. If the value of the attribute is '1', the data type will be returned with the corresponding information. Otherwise, the data type is not returned.

In the example below, taken from the Get PO transaction, the header information includes a requested PO number, and both the header and line information is requested to be returned.

```
<DOCUMENT>
    <GET_PO>
        <POHEADER returndata = "1">
            <POID>PO12345</POID>
        </POHEADER>
        <POLINE returndata = "1"/>
    </GET_PO>
</DOCUMENT>
```

Note that only the OAGIS data types use the **returndata** attribute. These are not used to select individual fields. It is assumed that all fields will be returned within the requested data type.

Also, data types requested for return, but not used for selection, do not need both start and end tags. In the example above, the **POLINE** element is implemented using a construct for indicating that the element has no additional content.

2.11 USERAREA Extensions

The USERAREA element of the OAGIS provides a facility for extending the transaction definitions to support additional data elements. These extensions may be required by software vendors, solution integrators, or customers. These constituencies have the same implementation requirements, and references to them will be made interchangeably.

Goals and Guidelines

The USERAREA should only be used in the rare occasion that specific data requirements are not supported by the transaction specification. Should a specific functional gap be identified in the specification, the Open Applications Group should be notified to consider including the required fields in the OAGIS.

As a part of the implementation design, a number of goals and guidelines were considered.

1. Allow multiple parties (vendors, integrators, or customers) to extend the same USERAREA.

   As a part of their standard product offerings, vendors may need to provide application specific elements within the USERAREA. In some cases, multiple vendors, or customers, may require extensions to the same USERAREA. The concurrent use of these extensions must not conflict with each other.

2. Support complex element definitions within each USERAREA.

   USERAREA implementations may be implemented with complex elements, including reuse of standard OAGIS segments. Any valid XML is supported within the USERAREA.

3. Element names must be unique within a transaction.

   Multiple USERAREA definitions are supported with a single transaction. Since each definition will be different, the vendor specific USERAREA elements must be uniquely identified. This allows element definitions to be encapsulated in a single vendor provided header file.

4. Element names must be unique across the specification.

   Multiple USERAREA definitions are supported across transactions. The vendor specific USERAREA definitions must be unique across transactions. However,

vendors may identify the need to reuse specific constructs across multiple transactions. This will allow for a single repository of vendor data elements for the entire set of supported transactions.

5. Changes required to the DTDs released by the Open Applications Group will be limited and isolated.

   Maintenance will be required as the OAGIS continues to be revised and expanded. The impact of this activity on vendor specific extensions should be limited as much as possible. This will be achieved by isolating the vendor specific information to DTD and message headers only. Extensions will not be embedded within the transaction files.

Naming Conventions

Naming conventions are established for each element in the USERAREA extension. These conventions reduce the likelihood of conflicts in element names, and provide consistency across implementations.

For illustration purposes, two fictional companies are used in the samples provided. They are Acme Software and ABC Widget.

TOP LEVEL USERAREA NAMING CONVENTIONS

A top level element will be defined for each vendor within each USERAREA extension. This element will be used to encapsulate the vendor's data elements. It provides both the context of the extension, as well as a reference to the vendor DTD to be used for validation.

The context of the extension is defined by the full element name path to the USERAREA. Each element within the path is separated by a period ('.').

Within each USERAREA, the top level element names will follow the format:

```
<vendor.transaction.context.USERAREA>
```

For example, the definition of the USERAREA extension in Acme Software's implementation of the JEHEADER element of the POST JOURNAL 004 transaction would be named:

```
<ACME.POST_JOURNAL_004.JEHEADER.USERAREA>
```

Similarly, ABC Widget's element would be named:

```
<ABC.POST_JOURNAL_004.JEHEADER.USERAREA>
```

Note that no data elements are defined at this level. The top level element provides context only and is used to encapsulate the data elements defined at the next level.

The definition of this element will be provided in the vendor provided DTD file. The vendor will provide supplemental DTDs that include only their proprietary extensions. The base DTD will be maintained and provided by the Open Applications Group.

ELEMENT NAMING CONVENTIONS

Within the top-level elements, vendor specific data elements are defined. These elements contain the required data, or definitions of complex data types.

Each element within the vendor-specific USERAREA definition will follow the format:

```
<vendor.elementname>
```

For example, the definition of the 'Weave' element within Acme Software's USERAREA would be named:

```
<ACME.WEAVE>
```

Reference Files

Each vendor will provide a single DTD that includes all the extensions required for an application/OAGIS release combination. Additional reference files may also be created by the integrators and customers as required for specific installations. Incorporating vendor extensions also requires minor changes to the Open Applications Group distributed reference files.

CUSTOMER/VENDOR REFERENCE FILES

Naming conventions for these files include Open Applications Group assigned vendor or customer prefixes, followed by optional product names and versions, or additional customer implementation information. In general, the file name will follow the format:

```
<vendor prefix>_<product>_<version>.dtd or
<customer prefix>_<implementation>.dtd
```

For example, Acme Software has released a DTD specifically for version 4.5 of their 'Planner' product. This file includes the element definitions for all the OAGIS transactions supported by their product. The DTD file name may be:

```
acme_planner_0405.dtd
```

Similarly, ABC Widget will also build a single DTD to support all of their proprietary extensions to the specification. These may be required for integration to legacy systems and to other products in their portfolio. Their DTD file name (for the US implementation) may be:

```
abc_usa.dtd
```

OPEN APPLICATIONS GROUP SPECIFICATION FILES

Minor changes to the released OAGI DTDs are required to support vendor extensions. These changes impact the field reference and each transaction DTD.

Field Reference: The definition of the USERAREA element is changed from a single string to support of any complex element. This change allows the vendor's top level elements to be used within the USERAREA element without specification changes to support each extension. This does not compromise the ability to use validating parsers, as long as no data elements are defined at the top level. The child data elements in the vendor extension are validated against the definition of the top level element in the vendor provided DTD.

This change is implemented in the Open Applications Group release of the field reference DTD.

Before:

```
<!ELEMENT USERAREA %STRDOM;>
```

After:

```
<!ELEMENT USERAREA ANY>
```

Vendor and Transaction DTD: Each vendor extension is incorporated into the transaction DTD. If multiple vendor extensions are required for an implementation, each one is added to the header of the transaction DTD.

This change is made at implementation time. As the system is installed, the implementation team will update the OAGI release files to include references to vendor extensions of each transaction required.

Add:

```
```

```
<!ENTITY % VENDOR SYSTEM "filename">
%VENDOR;
```

Example:

```
<!ENTITY % ACME_PLANNER_0405 SYSTEM "acme_planner_0405.dtd">
%ACME_PLANNER_0405;
```

Prefix Assignments

Vendors and customers (both members and non-members) may request a unique element prefix from the Open Applications Group. These will be uniquely defined, and must be periodically renewed.

While this registration is not required, the assignment of prefixes addresses the need for unique element tags within an implementation. Should an implementation incorporate prefixes that have not been assigned by the Open Applications Group, there is no guarantee that these will not conflict with implementations by other parties.

The prefix assignments will be used in the naming of both USERAREA 'wrapper' elements, as well as the data elements they contain. The naming conventions for these elements were described above.

For more information or specific prefix assignments for a product or customer, please contact the Open Applications Group.

*Execution Note: The vendor prefix assignments could be based on the registered domain name for the organization. In most cases, the *.com domain may be sufficient. For those organizations that do not have a North American corporate domain registered, the high level qualifier may also be required. For example, the prefix assigned to SAP may be either SAP or SAP-DE or SAP-COM. This subject will require more discussion to satisfy any vendor or solution provider needs.*

Examples

An example of a USERAREA implementation is provided here. For the purposes of illustration, extensions that may be desirable for an implementation at a fictional textile manufacturer are shown.

VENDOR DTD

Acme Software has developed an extension DTD for version 4.5 of their product. This is designed for use with version 6.0 of the OAGIS.

This file is named ACME_PLANNER_0405.dtd.

Within this file, a number of vendor specific elements are defined. These are related to additional information that is required for the product.

```
<!ELEMENT ACME.DYELOT (%STRDOM;)>
<!ELEMENT ACME.PARENTROLL (%STRDOM;)>
<!ELEMENT ACME.SYNTHETIC (%STRDOM;)>
<!ELEMENT ACME.THREAD (%STRDOM;)>
<!ELEMENT ACME.THREADCOUNT (%STRDOM;)>
<!ELEMENT ACME.TINT (%STRDOM;)>
<!ELEMENT ACME.WEAVE (%STRDOM;)>
```

After the elements are defined, they are incorporated into top level USERAREA elements.

For this example, the UPDATE INVENTORY and SYNC ITEM transactions are extended.

```
<!ELEMENT ACME.SYNC_ITEM_002.ITEMHEADER.USERAREA
  (ACME.THREAD, ACME.THREADCOUNT, ACME.SYNTHETIC?, ACME.WEAVE?,
   ACME.TINT?)>

<!ELEMENT ACME.UPDATE_INVENTORY_002.INVENTORY.USERAREA
  (ACME.DYELOT, ACME.PARENTROLL)>
```

CUSTOMER DTD

In addition to the extensions required for Acme Software, this implementation example also requires ABC Widget extensions for legacy system support. For this example, the project scope is limited to their American manufacturing plants.

This file is named ABC_USA.DTD.

Within this file, a number of customer specific elements are defined. These are related to additional information that is required for legacy integration.

```
<!ELEMENT ABC.BROKER (%STRDOM;)>
<!ELEMENT ABC.SKUGROUP (%STRDOM;)>
```

After the elements are defined, they are incorporated into top level USERAREA elements.

For this example, the SYNC ITEM transaction is extended.

```
<!ELEMENT ABC.SYNC_ITEM_002.ITEMHEADER.USERAREA
  (ABC.BROKER, ABC.SKUGROUP)>
```

XML FILES

The XML messages incorporate the vendor extensions within the USERAREA. Below is a section of the SYNC ITEM transaction, including the USERAREA extension.

```
<ITEMHEADER>
    . . .
    <ITEMDESC>CORD BLUE 2x50Y</ITEMDESC>
    <UPC>1254671703</UPC>
    <USERAREA>
        <ABC.SYNC_ITEM_002.ITEMHEADER.USERAREA>
            <ABC.BROKER>JAMAABC</ABC.BROKER>
            <ABC.SKUGROUP>CORD001</ABC.SKUGROUP>
        </ABC.SYNC_ITEM_002.ITEMHEADER.USERAREA>
        <ACME.SYNC_ITEM_002.ITEMHEADER.USERAREA>
            <ACME.THREAD>AA022</ACME.THREAD>
            <ACME.THREADCOUNT>120</ACME.THREADCOUNT>
        </ACME.SYNC_ITEM_002.ITEMHEADER.USERAREA>
    </USERAREA>
</ITEMHEADER>
```

2.12 Segment Extensions

The OAGIS contains complex data types, called Segments, which represent entities such as amounts or dates. Each segment contains multiple data elements. It also may have attributes, such as qualifiers or transaction types. A USERAREA implementation may incorporate Segments as a part of the extension definition. In some cases, a vendor or implementer requirement may require extending the list of valid qualifiers for a specific Segment. This section describes how Segment qualifiers lists are extended.

Segment Definitions

The XML definition of a Segment is shown below. It is structured in the same way as any other XML element. Usually, a Segment contains two attributes, the qualifier and type. Each attribute has a set of valid values that may be used.

In the example of the **OPERAMT** Segment provided below, the valid qualifier values are **'EXTENDED'** and **'UNIT'**. The valid type values are **'T'** and **'F'**. The remainder of the definition shows that the **OPERAMT** contains seven required elements.

```
<!ENTITY % SEG_OPERAMT_QUALIFIERS
    "(EXTENDED | UNIT)">
<!ENTITY % SEG_OPERAMT_TYPES
    "(T | F)">
<!ELEMENT OPERAMT (VALUE, NUMOFDEC, SIGN, CURRENCY, UOMVALUE,
  UOMNUMDEC, UOM)>
<!ATTLIST OPERAMT
    qualifier %SEG_OPERAMT_QUALIFIERS; #REQUIRED
    type %SEG_OPERAMT_TYPES; #REQUIRED>
```

In order to support extensions to the qualifier list, an additional qualifier, **"OTHER"**, is included in the default definition. This is incorporated as an expansion as shown below.

```
<!ENTITY % SEG_OPERAMT_QUALIFIERS_EXTENSION "OTHER">
<!ENTITY % SEG_OPERAMT_QUALIFIERS
    "(EXTENDED | UNIT | %SEG_OPERAMT_QUALIFIERS_EXTENSION;)">
```

At implementation time, the extensions provided by the vendor(s) replace the simple **"OTHER"** provided by default. The same naming conventions for vendor-specific elements are used for Segment extensions.

```
<!ENTITY % SEG_OPERAMT_QUALIFIERS_EXTENSION
    "OTHER | ACME.BATCH | ACME.LOT")
<!ENTITY % SEG_OPERAMT_QUALIFIERS
    "(EXTENDED | UNIT | %SEG_OPERAMT_QUALIFIERS_EXTENSION;)">
```

Should multiple extensions be required to support an implementation, the integrators will edit the Segments DTD to incorporate them into a single replacement entity.