HITIS Correlation and Interface Specification

Draft Standard

April 14, 2000

American Hotel & Motel Association
1201 New York Avenue
Suite 600
Washington, DC 20005

Prepared by:

American Hotel & Motel Association
1201 New York Avenue
Suite 600
Washington, DC 20005

www.ahma.com
www.hitis.org

Table of Contents

HITIS Correlation and Interface Specification

Interface Standard

HITIS

Correlation and Interface Specification

XML Interface Mapping

Draft Version 1.0

April 14, 2000

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Objects and XML

For the duration of the HITIS project, the sole mechanism for implementing the standards has been the use of Object Technology. Unfortunately, the use of the word "Objects" means different things to different people.

Perhaps the best way to communicate the concepts associated with the use of objects is to define some of the common terms in the Object environment.

The general understanding of those who deal with object technology is that everything is an object. The object can be either physical, such as a chair or a table, or virtual, such as a group of data that represents something. Objects, both real and virtual, can be categorized into classes. For the purposes of understanding the use of objects in the HITIS project, it is important to understand what is meant by the term "class" as it pertains to the object world.

Looking at furniture as an example, there are categories of furniture. Some furniture is made for sitting, some for sleeping, some for storage. The category of furniture could be called a class. You could have a class called "Chair" with all the furniture designed for sitting included. An instance of this class "Chair" might be an "Easy Chair" or "Stool". The specific chair is an object. The class is the category in which the chair fits.

In the case of data, a class is a definition of the data elements (known also as attributes or fields) which are part of a message. When data fills the data elements within that class, the result is an object. Systems in the past have taken these objects and placed the data into tables that are defined by the relationships among the data. This is known as a "Relational Database". In order to store the object, the data is broken into logical records (known as rows in a table) and stored with like rows. Each row is identified by some value that may be used to retrieve that row when the object is required. The separation of the storage schema and the object is the goal of object technology. Rather than pass data across processes, only pointers to the data are passed. This allows a system to maintain only one copy of the data, with multiple processes allowed access to the object through these pointers.

The elusive target for distributed computing has been that of distributed objects. Much discussion in the IT community has centered around the practicality of having disparate systems using remote operations. Remote systems performing operations on an object are given pointers to the object, or the object is encapsulated allowing no direct updating to the data. The inherent problem with this arrangement is that it requires a lot of data interchange when a remote system is performing operations on the object. This is not a problem in a Local Area Network (LAN) where the bandwidth is quite wide.

The attempts to implement HITIS standards using the Remote Object Invocation methodology within a Wide Area Network (WAN) uncovered what has been an ongoing flaw in this concept. Unless the data is actually transmitted, the network traffic required to resolve the data creates an unnecessary bottleneck. Passing the data in a packet can solve this problem. To do so, however, requires what is known as "serialization" of the object, that is, creating a packet that includes all the data. The class could be considered as the template for the packet, and that class or template filled with all the data becomes the object.

XML presents a simple way to represent the object in serialized form. By establishing the XML message format (effectively a class), the object can be formed into a packet and passed over

communication lines as an XML Document. There is no remote operation necessary, and the data has been passed from one system to another.

In the pure object environment, this would not be a desirable situation as it creates two versions of the data, and synchronization cannot easily be maintained. Within HITIS, however, this is not a problem as each document represents a transaction. Each transaction is purely a snapshot of the data (object). The transaction is sent from one system to another as a request for current data or to initiate an update of data on the remote system. In fact, one of the major aspects of an object oriented system is that updates are not done directly to data, but rather the data is encapsulated and transactions are sent to the system which houses the data, requesting the update to that data. The remote system does not do the updating, it only sends the data necessary to perform the update to the system housing the data.

Other transactions in the HITIS environment request status or a snapshot of the data as it exists at the moment the request is made. There is no reason for the separate systems to have access to the dynamic data. In fact, when communicating between systems in an object environment, it is usually necessary for the system that houses the object to make a copy of that object for the use of the remote system. If this is not done, the data may change while the remote system is referencing that data, and as a result, pointers would change.

The net result of implementation under XML is that the HITIS standards that once consisted of operations have been reformatted into messages rather than object-based operations. The parameters of the operations were transformed into the data of the message, and most operations resulted in a pair of messages. One message initiates a request, with a corresponding message responding. The initiating system may be sending a request to update data on the remote system, with the remote system responding with the status of that transaction. In another scenario, the initiating system may be requesting data, with the responding system returning the data.

1.2 The Use of XML Schema

A considerable amount of work within the HITIS committees went in to the identification and standardization of data types for use within the HITIS standards.

Because of the differences in the way various operating systems handle dates, time, and floating point data, it was determined that there should be a clearly defined set of data types which would be used consistently within all standards. Some of these data types are native to all systems, such as integers, while others are derived from more basic data types.

An example of the derived data type would be a “datetime”, where the Year, Month, Day, Hour, Minute, Seconds, and offset from UTC (also known as GMT) are used to represent a specific time. Each of the sub-elements consists of integers, with all but the offset specified as positive (unsigned) integers only. The offset may be a negative number, therefore it must be a signed integer.

XML presents an additional issue, in that the data within an XML document can only be string data. With an XML implementation it is necessary to agree on the format in which anything other than a string would be represented in the string only data. The World Wide Web Consortium (W3C) recommendation for XML utilizes **Document Type Definitions** (DTD's) as a way of identifying the data element names that are permitted within a document, but not in identifying the data types of those elements.

One of the emerging XML standards resolves this issue by using a standardized format for all non-string data and by allowing each data element to be identified as to the type of data which is

represented. This mechanism is known as **XML Schema**. While the specification of the W3C has not been finalized, there is a great deal of momentum behind this initiative, and, although there is more than one organization involved in the definition of XML Schema, there is much commonality in what is being proposed. As the issues are resolved, there is no doubt that XML Schema will replace the Document Type Definition as the validation mechanism for an XML document.

One of the strong points of XML Schema is that the schema document is actually an XML document, unlike a DTD, which has its own format. XML Schema also allows the specification of a data type for an element or attribute. Data types specify the format of the data, provide for validation of the type by the XML parser, and enable data-type-specific processing. IBM, Microsoft, and Sun Microsystems all endorse the proposed XML Schema recommendation. There would be little risk in using the XML Schema method as the single way of documenting the HITIS message format.

1.3 Infrastructure issues

While the task of HITIS is to define the message sets which are to be used to transfer data from one system to another, there is a higher level of infrastructure which must be addressed. Specifically, the issues of identifying the sender, the receiver, the message type, and other non message specific data should not be the problem of the HITIS technical committees.

Among the industry standard initiatives that have the responsibility to create a common infrastructure, one is XML.org, an open group to which IBM, Sun, and Microsoft all subscribe. This organization has the capability to become the repository for the XML Schema or DTD and the keeper of the infrastructure.

The other initiative sponsored by Microsoft is known as BizTalk. BizTalk creates an infrastructure within which an XML Document can be enclosed, and which takes care of all the identification and security issues. BizTalk also provides a repository in which DTDs or XML Schema documents may be posted on behalf of an organization presenting standards to its industry.

The HITIS standards may be enabled for posting at both BizTalk and XML.org sites. The BizTalk environment requires that a document begin with the <BizTalk> tag and end with the </BizTalk> tag. If these can be optional, then any vendor wishing to use BizTalk (most likely in Microsoft development tools) could do so with no problems. Those not using BizTalk would have to resolve the infrastructure issues amongst themselves.

It is the intent of HITIS that an XML message instance could be passed and validated to a BizTalk schema and to an XML.org (W3C) schema. This would require using XML-Data data types or declaring any HITIS-defined data types in the schema definition. Irrespective of the process of validation, it is unclear whether using the same implied tags provided by BizTalk would allow a system using BizTalk and one using XML.org infrastructure to interoperate.

2.0 SUMMARY

This document describes the conventions used in the HITIS object specifications and in the XML interface protocol. Specifically, it defines the fundamental data types and maps them to the XML Schema Definition Language as proposed by the W3C. The object data types, including the ranges of values, and the objects and attributes common to all HITIS interfaces are described along with their correlating representation in eXtensible Markup Language, XML. In addition, the conventions for mapping all HITIS object operations to message sets are described, including rules for naming certain corresponding elements and attributes in XML.

3.0 DATA TYPES

3.1 Integer and Unsigned Integer

Object Value:
Signed Integers (Integers) and Unsigned Integers (UInts) are 32-bit numeric non-fractional values. Integers range from –2^31 to 2^31-1, or approx. -2 billion to +2 billion. Unsigned Integers range from 0 to 2^32-1 or 0 to approx. 4 billion. (U.S. billions--1,000,000,000 -- not U.K. billions.)

XML Representation:
XML representation of the Signed Integer and Unsigned Integer (Uint) data types would use the same data type(s) as defined by the emerging W3C specification, with the unsigned Integer using the non- positive / negative integer values as defined below:

[W3C Definition - W3C Working Draft 25 February 2000]

The integer datatype derived from decimal by fixing the value of scale to be 0. This results in the standard mathematical concept of the integer numbers. The value space of the integer datatype is the infinite set {...,-2,-1,0,1,2,...} The basetype of integer is decimal.

Integer has the following constraining facets:

maxInclusive
maxExclusive
minInclusive
minExclusive

integer has the following subtypes:

non-negative-integer
non-positive-integer

Lexical representation

Integer values have a single, standard lexical representation. This consists of a string of digits with an optional leading sign. If the sign is omitted, "+" is assumed. For example: -1, 0, 12678967543233, +100000.

non-negative-integer

[Definition:] The non-negative-integer datatype is the standard mathematical concept of the non-negative integers. The value space of the non-negative-integer datatype is the infinite set {0,1,2,...,∞} although computer implementations restrict this to a finite set. The basetype of integer is integer.

non-negative-integer has the following constraining facets:

maxInclusive
maxExclusive
minInclusive
minExclusive

non-negative-integer has the following subtypes:

positive-integer

Lexical representation

Non-negative-integer values have a single, standard lexical representation. This consists of a string of digits with an optional leading "+" sign. If the sign is omitted, "+" is assumed. For example: 1, 0, 12678967543233, +100000.

positive-integer

[Definition:] The positive-integer datatype is the standard mathematical concept of the positive integers. The value space of the positive-integer datatype is the infinite set {1,2,...,∞} although computer implementations restrict this to a finite set. The basetype of integer is non-negative-integer.

positive-integer has the following constraining facets:

maxInclusive
maxExclusive
minInclusive
minExclusive

Lexical representation

positive-integer values have a single, standard lexical representation. This consists of a string of digits with an optional leading "+" sign. For example: 1, 12678967543233, +100000.

non-positive-integer

[Definition:] The non-positive-integer datatype is the standard mathematical concept of the non-positive integers. The value space of the non-positive-integer datatype is the infinite set {-∞,...,-2,-1,0} although computer implementations restrict this to a finite set. The basetype of integer is integer.

non-positive-integer has the following constraining facets:

maxInclusive
maxExclusive
minInclusive
minExclusive

non-positive-integer has the following subtypes:

negative-integer

Lexical representation

Non-positive-integer values have a single, standard lexical representation. This consists of a string of digits with a leading "-" sign. For example: -1, 0, -12678967543233, -100000.

negative-integer

[Definition:] The negative-integer datatype is the standard mathematical concept of the negative integers. The value space of the negative-integer datatype is the infinite set {-∞,...,-2,-1}

although computer implementations restrict this to a finite set. The basetype of integer is non-positive-integer.

negative-integer has the following constraining facets:

maxInclusive
maxExclusive
minInclusive
minExclusive

Lexical representation

negative-integer values have a single, standard lexical representation. This consists of a string of digits with a leading "-" sign. For example: -1, -12678967543233, -100000.

3.2 Long and Unsigned Long

Object Value:
Signed Longs (Longs) and Unsigned Longs (Ulongs) are 64-bit numeric non-fractional values. Longs range from –2^63 to 2^63-1, or approx. -9 quintillion to +9 quintillion. Unsigned Longs range from 0 to 2^64-1, or approx. 18 quintillion. (U.S. value, not U.K.)

XML Representation:
XML representation of the Signed Long and Unsigned Long data types would use the same data type(s) as used for (2.1) Integer and Unsigned Integer, with the unsigned Long using the positive / non-negative integer values as defined above.

3.3 String

Object Value:
Strings are a UNICODE collection of characters and have no maximum length. HITIS specifies the length of a string that is the minimum length required for persistence that each compliant system must be capable of saving. The type of strings used are mapped to the specific platform, and it is assumed that the underlying object architecture or the implementation, if required, somehow tags strings with their length before attempting to pass them across process boundaries. If a greater length than the minimum required for persistence is used, there is no guarantee that the other system will save the greater length, therefore, truncation of extra characters by the receiving system (e.g., when storing the string in a database) is not an error. A string is Immutable once the data has been passed to the receiving system, as the character collection is considered a snapshot from the sending system.

XML Representation:
XML representation of the string data type would be the same data type defined by the emerging W3C specification, with the constraints as defined below:

[W3C Definition - W3C Working Draft 25 February 2000]

The **string** datatype represents character strings in XML. The value space of the string datatype is the set of finite sequences of UCS characters ([ISO10646] and [Unicode]). A UCS character (character, for short) is an atomic unit of communication; it is not further specified except to note that every UCS character has a corresponding UCS code point, which is an integer. The ordered property of **string** is the [Unicode] character number sequence.

The string datatype has an optional constraining facet called **pattern**. The value of this facet is a **regular expression**. A regular expression is an alphanumeric string consisting of character symbols. Each symbol, which is usually one character but may be two characters, is a placeholder that stands for a set of characters. If this facet is not present, there is no restriction on the lexical representation.

The string datatype has an optional constraining facet called **length**. For the string datatype, **length** specifies the number of allowable characters in the string. If length is specified, the data type is a fixed length character string, where length is measured in the number of characters in the string. If length is not specified, it is a variable length character string. The value of the length facet must be a positive integer.

The string datatype has an optional constraining facet called **maximum length**. If maxlength is specified for a variable length string it represents an upper bound of the length of the string. The value of the maxlength facet must be a positive integer. Both length and maximum length cannot be specified for the same datatype. The absolute maximum length of variable length character strings depends on the XML parser implementation.

Maximum and Minimum Values - The string datatype also has the following constraining facets:

maxInclusive

maxExclusive

minInclusive

minExclusive

Clearly, the effect of these constraining facets depends on the collating sequence used to define the **order** property for strings. A value space, and hence a datatype, is said to be **ordered** if there exists an **order relation** defined for that value space. Order relations have the following rules:

for every pair (a, b) from the value space, either $a \leq b$ or $b \leq a$, or $a = b$;

for every triple (a, b, c) from the value space, if $a \leq b$ and $b \leq c$, then $a \leq c$.

There may exist several possible order relations for a given value space. Additionally, there may exist multiple datatypes with the same value space. In such cases, each datatype will define a different order relation on the value space.

There is no XML representation for the minimum length and an unlimited maximum of a string, therefore, HITIS cannot require a length for persistence, but suggests that the minimum length recommended in the object specifications be able to be reconstituted for passing the data on to another system.

3.4 DateTime

Object Value:

Datetime values are derived data types using unsigned integers. DateTime is defined as eight (8) parts: Year, Month, Day, Hour, Minute, Second, TimeZone and TimeType. (NOTE: COM implements DateTime using 2 Longs. COM dates have a resolution of approx. 1 millisecond and range from January 1, 100, to December 31, 9999.) The domain of year = 4 digits numeric.

TimeZone (GMT or UTC) is a separate attribute that modifies DateTime. TimeZone is a signed attribute, defined as indicating +- 12 hours from GMT, and has a domain of values.

TimeType is a HITIS enumerated type with the values: 1 = time relevant, 2 = timeNOT relevant, 3 = local time.

XML Representation:

The emerging W3C specification defines an XML representation of the dateTime data type as a timeInstant with the following constraints:

[W3C Definition - W3C Working Draft 25 February 2000]

The **timeInstant** datatype represents a combination of date and time values representing a single instant of time. The value spaceof **timeInstant** is the space of Gregorian dates and legal time values as defined by [ISO 8601].

A single lexical representation, which is a subset of the lexical representations allowed by ISO 8601 is allowed for **timeInstant**. This lexical representation is the ISO 8601 extended format CCYY-MM-DDThh:mm:ss.sss where "CC" represents the century, "YY" the year, "MM" the month and "DD" the day, preceded by an optional leading sign to indicate a negative number. If the sign is omitted, "+" is assumed. The letter "T" is the date/time separator and "hh", "mm", "ss.sss" represent hour, minute and second respectively. Additional digits can be used to increase the precision of fractional seconds if desired. To accommodate year values greater than 9999 additional digits can be added to the left of this representation.

This representation can be immediately followed by a "Z" to indicate Coordinated Universal Time. To indicate the time zone, i.e. the difference between the local time and Coordinated Universal Time, the difference immediately follows the time and consists of a sign, + or -, followed by hhmm.

For example, to indicate 1:20 pm on May the 31st, 1999 for Eastern Standard Time which is 5 hours behind Coordinated Universal Time, one would write: **1999-05-31T13:20:00-05:00**.

Variations in the use of the ISO 8601 allow for a **date**, **time**, and **time.tz**, as well as **dateTime** and **dateTime.tz** data types that can be read and understood by XML parsers. It can be assumed that if a date is used without a time in a message, that the time is NOT relevant, and that if a full dateTime value is sent, that time is relevant.

The inclusion of the time zone indicates a value of time in relation to UTC, but the absence of the time zone portion can be assumed to be local time. This assumption allows for elimination of the TimeType attribute in the XML mapping. In most cases, the HITIS data types for an attribute designate the **DateTime** data type, but certain instances may find it appropriate to use any of the following allowable combinations. e.g.: the PostingDate attribute which is a hotel fiscal date and does not require a time.

date	Date in a subset ISO 8601 format, without the time data. For example: "1994-11-05".
dateTime	Date in a subset of ISO 8601 format, with optional time and no optional zone. Fractional seconds can be as precise as nanoseconds. For example, "1988-04-07T18:39:09".
dateTime.tz	Date in a subset ISO 8601 format, with optional time and optional zone. Fractional seconds can be as precise as nanoseconds. For example: "1988-04-07T18:39:09-08:00".
time	Time in a subset ISO 8601 format, with no date and no time zone. For example: "08:15:27".
time.tz	Time in a subset ISO 8601 format, with no date but optional time zone. For example: "08:1527-05:00".

3.5 Currency

Object Value:

Currency (money value) is an object with two attributes: CurrencyCode, a 3-character string indicating the ISO currency code (e.g., “USD”, “CAD”, etc.) and Amount, which is derived from the Long data type. Amount values are scaled by 1 million to accommodate the current European Union standard. The currency data type has a range of (minus) -2^63/1000000 to +(2^63-1)/1000000, or approximately -9 trillion to +9 trillion (U.S. trillions, not U.K.). The currency code is ISO standard # 4217 (1995). See *References*.

XML Representation:
HITIS maps the Currency object value to XML by adhering to the two (2) attributes of the object model. Any element that is designated with the Currency data type will have a CurrencyCode attribute, and an Amount sub-element whose data type is a decimal.

The HITIS Currency data type is expressed in the following example of a FolioBalance:

```
<FolioBalance CurrencyCode="USD">
    <Amount>14.32</Amount>
</FolioBalance>
```

The DTD for an element using the HITIS Currency data type is:

```
<!ELEMENT Currency (Amount )>
<!ATTLIST Currency CurrencyCode CDATA  #REQUIRED>
<!ELEMENT Amount (#PCDATA )>
<!ATTLIST Amount e-dtype NMTOKEN  #FIXED 'decimal'>
```

The XML Schema definition is:

```
<!-- Conforms to w3c http://www.w3.org/TR/xmlschema-1/-->
<schema targetNamespace="Currency.dtd" xmlns="http://www.w3.org/1999/05/06-xmlschema-1/structures.xsd">
    <element name="Currency">
        <type content="elementOnly">
            <group order="seq">
                <element ref="Amount"/>
            </group>
            <attribute name="CurrencyCode" minOccurs="1" type="string"/>
        </type>
    </element>
    <element name="Amount" type="decimal"/>
</schema>
```

NOTE:
A common alternative representation of the **currency** data type is a fixed decimal 14.4.

fixed.14.4	Same as "number" but no more than 14 digits to the left of the decimal point, and no more than 4 to the right.

The fixed 14.4 currency data type is not used in the HITIS specifications.
Exception: If the fixed 14.4 Currency data type is adopted as part of a future W3C specification, HITIS will defer to this standards body definition.

3.6 Percent

Object Value:
Percent is a data type derived from the Long data type (Signed Long). Percent data can represent positive or negative fractional (less than 1) values. Values are scaled by 1 million to accommodate 4 decimal places, plus the two for the percentage. (For example, 10% would be expressed as 100000; 0.5% would be expressed as 5000.) Thus, the percentage data type has a range of -2^63/10000 to +(2^63-1)/10000, or approximately -900 trillion percent to +900 trillion percent. (U.S. trillions, not U.K.)

XML Representation:

There is no direct XML equivalent of **percent**. It is necessary to use a data type from the existing base data type of *integer* (a *decimal*). A decimal is expressed as a true percentage in lexical representation; e.g.: 1% is expressed as 0.01, and 10% is expressed as 0.10, reflecting its actual decimal value.

The emerging W3C specification defines an XML representation of a built-in generated data type of the **decimal** data type as follows:

[W3C Definition - W3C Working Draft 25 February 2000]

decimal represents arbitrary precision decimal numbers. The value space of **decimal** consists of the values *i* × *10^n*, where *i* and *n* are integers, with *n* being known as the scale of the value space.

decimal has a single standard lexical representation. This consists of a finite sequence of decimal digits separated by a period as a decimal indicator, in accordance with the scale and precision facets, with an optional leading sign. If the sign is omitted, "+" is assumed. Leading and trailing zeroes are optional. For example: -1.23, 12678967.543233, +100000.00.

Decimal has the following constraining facets:

precision
scale
pattern
enumeration
maxInclusive
maxExclusive
minInclusive
minExclusive

decimal has the following built-in datatypes:

integer

Decimal values have a single standard lexical representation. This consists of a string of digits separated by a period as a decimal indicator, in accordance with the scale and precision facets, with an optional leading sign to indicate a negative number. If the sign is omitted, "+" is assumed. Leading and trailing zeroes are optional. For example: -1.23, 12678967.543233, 100000.00.

The DTD and XML-Schema definition for elements using the data type *decimal* is:

```
<!ELEMENT CommissionPercent (#PCDATA )>
<!ATTLIST CommissionPercent e-dtype NMTOKEN  #FIXED 'decimal' >

<schema xmlns="http://www.w3.org/1999/05/06-xmlschema-1/structures.xsd">
    <element name="CommissionPercent" type="decimal"/>
</schema>
```

3.7 Byte Array

Object Value:

A Byte Array is an array of bytes (8-bit unsigned numeric non-fractional value), and is used for vendor-specific extensions (i.e., additions) of complex data to certain objects. For example, a guest profile may include a bitmap image of a photograph. A byte array is **immutable**, as the data cannot be appended once it is sent to the other system. The data passed is considered a snapshot from the sending system. Byte arrays are implemented however the underlying object architecture implements them, and a system that does not accommodate the data format received is tasked with mapping the raw binary data to a data type that the system uses.

XML Representation (W3C):
The emerging W3C specification defines an XML representation of a primitive data type, **binary,** as follows:

[W3C Definition - W3C Working Draft 25 February 2000]

binary represents arbitrary binary data. The value space of **binary** is the set of finite sequences of binary octets.
Constraint: encoding required for binary
It is an error for **binary** to be used directly in a schema. Only datatypes that are derived from **binary** by specifying a value for encoding can be used in a schema.
binary has the following constraining facets:

encoding
length
minlength
maxlength
pattern
enumeration

If the length is not specified, a datatype with variable length is specified. In this case, the optional maximum length facet specifies the maximum length of the data in bits. If the maximum length is not specified the default is unlimited length. The optional "encoding" facet specifies the encoding which may be "**hex**" for hexadecimal digits or "**base64**" for MIME style Base64 data.

The HITIS DTD and XML-Schema definition files will indicate a **bin.base64** datatype for elements designated as a Byte Array in the object model. In order to enable a programmer to parse out all CDATA without validation and map the raw binary data format received in way that their individual system can accommodate it, these variables may be included as CDATA in an XML message instance.

NOTE
XML schema generation tools provide for a **bin.base64,** or **bin.hex** data type that corresponds to the binary data type in the specification above. The HITIS standards have designated use of the **bin.base64** data type for representing binary data passed in conjunction with an XML message assuming XML processors support conversions between these types.

bin.base64	MIME-style Base64 encoded binary BLOB.
bin.hex	Hexadecimal digits representing octets.

The following example shows the use of the binary data type in the ContentData element (a part of the Extension object):

```
<Extension>
    <ContentData>3456.jpg</ContentData>
</Extension>
```

The DTD and XML-Schema definition for elements using the data type *bin.base64* is:

```
<!ELEMENT ContentData (#PCDATA )>
<!ATTLIST ContentData e-dtype NMTOKEN  #FIXED 'bin.base64'>

<schema  xmlns="http://www.w3.org/1999/05/06-xmlschema-1/structures.xsd">
   <element name="Extension">
        <type content="elementOnly">
                <element ref="ContentData"/>
        </type>
```

```
    </element>
    <element name="ContentData" type="bin.base64"/>
</schema>
```

3.8 Boolean

Object Value:
Boolean is a binary value with a domain of zero (0) or one (1). Allowable values are 0, which indicates False or No, and 1, which indicates True or Yes. A Boolean value will be mapped to the environment and implemented however the underlying architecture defines it. A bridge may be required to map data type discrepancies between platforms.

XML Representation (W3C):
The emerging W3C specification defines an XML representation of a primitive data type, **boolean,** as follows:

> [W3C Definition - W3C Working Draft 25 February 2000]
>
> **boolean** has the value space required to support the mathematical concept of binary-valued logic: {true, false}.
>
> An instance of a datatype that is defined as *boolean* can have the following legal lexical values {true, 1, false, 0} with '1' being the same as 'true' and '0' being the same as 'false'.
>
> **boolean** has the following constraining facets:
> pattern

The HITIS standards use the **boolean** data type that corresponds to the W3C XML specification.

boolean 0 or 1, where 0 == "false" and 1 =="true".

An example of the use of the **boolean** data type in the IsCredit element (part of the QueryFolioRequest message) is as follows:

```
<IsCredit>true</IsCredit>
```

The DTD for this element would be:

```
<!ELEMENT IsCredit (#PCDATA )>
<!ATTLIST IsCredit e-dtype NMTOKEN  #FIXED 'boolean'>
```

The XML-schema definition for this element would be:

```
<Schema name="QueryFolioRequest.dtd" >
    <ElementType name="IsCredit" content="textOnly" dt:type="boolean"/>
</Schema>
```

3.9 DayOfWeek (DOWPattern)

Object Value:
The Day of Week data type (DOWPattern) is an object with 7 Boolean attributes. i.e.: Sunday, Yes/No; Monday, Yes/No; Tuesday, Wednesday, Thursday, Friday, and Saturday, etc.

XML Representation:
There is no XML equivalent for this HITIS-defined data type. An XML mapping is created from the existing base data type of **boolean**.

Mapping the HITIS object value to XML following strict adherence to the seven (7) attributes of the object model results in a **DOWPattern** value as follows: [a sample use of DOWPattern in a DaysOfWeek element (part of the RateAmount object)]

```
<DaysOfWeek>
    <DOWPattern>
        <Monday>true</Monday>
        <Tuesday>true</Tuesday>
        <Wednesday>true</Wednesday>
        <Thursday>true</Thursday>
        <Friday>false</Friday>
        <Saturday>false</Saturday>
        <Sunday>false</Sunday>
    </DOWPattern>
</DaysOfWeek>
```

The DTD for the DOW Pattern is:

```
<!ELEMENT DOWPattern (Monday , Tuesday , Wednesday , Thursday , Friday , Saturday , Sunday , DaysOfWeek )>
<!ELEMENT Monday (#PCDATA )>
<!ATTLIST Monday e-dtype NMTOKEN  #FIXED 'boolean'>
<!ELEMENT Tuesday (#PCDATA )>
<!ATTLIST Tuesday e-dtype NMTOKEN  #FIXED 'boolean'>
<!ELEMENT Wednesday (#PCDATA )>
<!ATTLIST Wednesday e-dtype NMTOKEN  #FIXED 'boolean'>
<!ELEMENT Thursday (#PCDATA )>
<!ATTLIST Thursday e-dtype NMTOKEN  #FIXED 'boolean'>
<!ELEMENT Friday (#PCDATA )>
<!ATTLIST Friday e-dtype NMTOKEN  #FIXED 'boolean'>
<!ELEMENT Saturday (#PCDATA )>
<!ATTLIST Saturday e-dtype NMTOKEN  #FIXED 'boolean'>
<!ELEMENT Sunday (#PCDATA )>
<!ATTLIST Sunday e-dtype NMTOKEN  #FIXED 'boolean'>
```

The XML-Schema definition for the DOW Pattern is:

```
<schema targetNamespace="DOWPattern.xsd" xmlns="http://www.w3.org/1999/05/06-xmlschema-1/structures.xsd">
    <element name="DOWPattern">
        <type content="elementOnly">
            <group order="seq">
                <element ref="Monday"/>
                <element ref="Tuesday"/>
                <element ref="Wednesday"/>
                <element ref="Thursday"/>
                <element ref="Friday"/>
                <element ref="Saturday"/>
                <element ref="Sunday"/>
            </group>
        </type>
    </element>
    <element name="Monday" type="boolean"/>
    <element name="Tuesday" type="boolean"/>
    <element name="Wednesday" type="boolean"/>
    <element name="Thursday" type="boolean"/>
    <element name="Friday" type="boolean"/>
    <element name="Saturday" type="boolean"/>
    <element name="Sunday" type="boolean"/>
</schema>
```

3.10 DateTimeSpan

Object Value:

The DateTimeSpan data type is an object with 3 attributes: StartTime[1]: DateTime, TimeUnit: TimeUnitType, and NumberOfTimeUnits: Integer.

XML Representation:
There is no XML equivalent of **DateTimeSpan**. A HITIS data type is created from the existing base data type of *timeDuration*.

> [Definition:] The **timeDuration** datatype represents a combination of year, month, day and time values representing a single duration of time, encoded as a single string. A single lexical representation, which is a subset of the lexical representations allowed by [ISO 8601], is allowed for timeDuration.
>
> Lexical Representation
>
> The lexical representation for timeDuration is the [ISO 8601] representation CCYYMMDDThhmmss.sss, preceded by an optional sign (+ or -), where "CC" represents the number of centuries, "YY" the number of years, "MM" the number of months and "DD" the number of days. The letter "T" is the date/time separator and "hh", "mm", "ss.sss" represent number of hours, minutes and seconds respectively. Note that this representation allows for fractional seconds.
>
> For example, to indicate a duration of 1 year, 2 months, 3 days, 10 hours, and 30 minutes, one would write: 00010203T103000.
>
> Time periods, i.e. specific durations of time, can be represented by supplying two items of information: a start instant and a duration or a start instant and an end instant or an end instant and a duration.

The HITIS standards use the XML mapping that provides for two elements to represent the specific period of time; a startInstant and a duration. The lexical representation of the **dateTime** data type defined by [ISO 8601] is used in both elements to build the **DateTimeSpan** data type.

The following is an example of the XML mapping of a DateTimeSpan:

```
<DateTimeSpan>
    <startInstant>1999-05-31T13:20:00</startInstant>
    <duration>+00000002T474000</duration>
</DateTimeSpan>
```

The DTD for the HITIS DateTimeSpan is:

```
<!ELEMENT DateTimeSpan (startInstant , duration )>
<!ELEMENT startInstant (#PCDATA )>
<!ATTLIST startInstant e-dtype NMTOKEN  #FIXED timeInstant>
<!ELEMENT duration (#PCDATA )>
<!ATTLIST duration e-dtype NMTOKEN  #FIXED timeInstant >
```

The XML-schema definition for the HITIS DateTimeSpan is:

```
<schema targetNamespace="DateTimeSpan.dtd" xmlns="http://www.w3.org/1999/xmlschema/datatypes">
    <element name="DateTimeSpan">
        <type content="elementOnly">
            <group order="seq">
                <element ref="startInstant"/>
                <element ref="duration"/>
            </group>
        </type>
    </element>
    <element name="startInstant" type="timeInstant"/>
```

[1] The attribute Start was renamed StartTime to avoid conflicts with the word Start, which is reserved in some programming languages.

```
    <element name="duration" type="timeInstant"/>
</schema>
```

4.0 SESSION OBJECT

Object Values:

The Session object represents the connection to a server. Note that this object is the only one directly creatable (i.e., instantiable) by a client application. All other objects in a HITIS interface are created via object operations (or accessed via object attributes). Therefore, besides the standard attributes and operations described below, the Session object for each HITIS interface may have additional attributes and/or operations to access or create other objects.

XML Representation

In converting the UML model to a messaging format, the object-based operations were removed, converting the business functionality they represented into request/response pairs. A Request message contains the input parameters of the object-based operation, and a Response message contains the return or *[Out]* parameter(s), if any, and the HITIS Code attribute that indicates the status of the processing of the message by the receiving system, or an error condition, if one exists.

4.1 Versioning

Object Values:

The session object for each interface should be named “xSession”, where "x" indicates the name of the interface, e.g., “SecuritySession” for the Security interface’s session object. The following attributes are included in every Session object:

4.1.1 VersionMajor (Integer)

Defines the Major version of the HITIS interface, indicated in decimal notation by the integer preceding the decimal. e.g.:: Version 1.1

4.1.2 VersionMinor (Integer)

Defines the Minor version of the HITIS interface, indicated in decimal notation by the integer following the decimal. eg: Version 1.1

XML Representation

The version numbering scheme remains the same in XML, combining both major and minor version in the element tag <Version>. The data type will be a **string**, thus allowing the use of letters in combination with whole integers for interim sub-versions, e.g.: 1.1c.

4.2 Operations

Object Values:

The following operations are included in every Session object:

4.2.1 Authenticate (UserName : String, Password : String) : Integer

The Authenticate operation is used to log in to the server and authenticates the user. The location of the object is determined outside the operation when the application originally creates the Session object.

Exception - The Authenticate operation in the Event Notification object is different from the other standards because it defines an asynchronous event and contains an additional parameter for the event's Sequence Number.

Authenticate (UserName : String, Password : String, ServerSequenceNumber : Integer) : Integer

4.2.2 GetErrorString

The GetErrorString operation is a response to an error condition that has occurred in the interface object. The operation provides a text description of the error associated with the ErrorNumber and may supply that text in a selected language, if that is an available feature of the application.

GetErrorString (ErrorNumber : Integer, LanguageID : String, ErrorString : String) : Integer

Vendors can map to HITIS string definitions or override them with their own string as vendors are responsible for having a string that describes each error number returned. There is no length restriction on error strings, and error stings are read-only.

4.2.3 LanguageID

Object Values:

The default language for HITIS-defined error strings is English, but the error text may be supplied in a selected language, if that is an available feature of the application processing the messages. If a translation is provided for the string, the LanguageID element is filled out to specify the language of translation. If a match is not found for the error number in the LanguageID requested, then the default LanguageID (English) is returned.

The LanguageID attribute is a combination of two strings; Language and CountryCode. The 1st string designates the Language, the 2nd string is the CountryCode; the delimiter is an underscore. For example, **en_UK**, (English, United Kingdom). The ISO standard for Language is 639:1988 and the ISO standard for Country Codes is 3166-1:1997. See *References.*

XML Representation

NOTE* - The revised, February 25, 2000, W3C specification defines a derived datatype, **language,** as follows:

> [Definition:] **language** represents natural language identifiers as defined by[RFC 1766]. The value space of **language** is the set of all strings that match the LanguageID production in [XML 1.0 Recommendation]. The lexical space of **language** is the set of all strings that match the LanguageID production in [XML 1.0 Recommendation]. The base type of **language** is string.

H. Alvestrand, ed. *RFC 1766: Tags for the Identification of Languages* 1995. Available at: http://www.ietf.org/rfc/rfc1766.txt

XML Representation

The Authenticate and GetErrorString operations of the Session object are replaced by the XML messaging paradigm, allowing for three distinct message types, in a specific request/response message pair that defines how the exchange of messages will take place.

The HITIS standards define three types of messages that a system can perform at the infrastructure level, and are separate from the business context.

1. Register/UnRegister - Initial agreement to decide how to exchange data
2. Subscribe/ UnSubcribe - Request to receive all published events of a certain type
3. HITIS Message - Initiation of all business messages (Request and Response)

HITIS message types contain a Header and a Body. The Header is identical in all message types. The LanguageID that determines the administration and error string language is used in the HITISRegister message.

5.0 HITIS HEADER

A HITIS XML document should be able to be routed through a system and traceable by a logfile. The HITIS Header section is designed to provide sufficient information to correlate a request/ response to the message and to enable settlement between business partners. The HITIS message may be encapsulated in a wrapper that handles the routing in the infrastructure, but if a document becomes separated, the Header is designed to allow the transport protocol to route the message appropriately.

The Header has two attributes:

- **Original Body Requested** - a True/False Boolean that indicates "Echo reply". Some systems save the content of the original message, but other systems cannot process the response unless the original request is also returned. If OriginalBodyRequested = true, the requestor should put the whole XML request and reply together in the DOM to return them in the response message. Returning the original body is optional in the HITIS standards and depends on the requesting system.
- **ImmediateResponseRequired** - a True/False Boolean that indicates a synchronous message. This communicates the necessity to hold the thread open at the switch for the response to be returned along with the acknowledgement in *http*. In the absence of using *http*, use of this attribute indicates whether a thread is blocking or non-blocking despite the underlying protocol.

5.1 Routing Elements

The FromURI, and ToURI elements are system addresses that identify the origin and destination of the message. These elements record the systems (business partners) that a message passes through on its way to the system that will process the request. Whenever a HITIS message is forwarded to another system, the new path is appended by the router. The "To" replaces the "From" and adds a new destination. The ReplyToURI is the final location address that anticipates the business response.

The "To" and "From" URIs are business entities that a message passes through and are recorded in the Routing Hops element for reservation transactions, and persisted in the Reservation record. The Routing Hops are used as a reference to handle the ownership of business rules and determine the business agreement or permissions allowed between trading partners. Routing hops may also affect booking rules, such as accepting a reservation under certain conditions that are different from accepting that reservation from another system. If a reservation changes they are used to synchronize all systems that need to be notified. They are also used to avoid circular references.

The transmission path of a message is not necessarily relevant to the business context of a message. A message may be routed by a switch that determines the most efficient path for delivery, and the physical routing is handled by the transport layer.

5.2 Message IDs

A Message ID is created by the source and is used to identify the original message. The Message ID defines the message set of a request and response pair. A message may be queued, asynchronous of the context of the message, and systems can use the MessageID to match up the result with the request. If a message needs to be re-posted, a Message ID is also used in the re-post.

A MessageID is filled in by each sender (the <FromURI>). The <Original Message ID > is a means by which to identify the original message. The originator of the message populates both fields, the MessageID and the Original MessageID.

5.3 Token

A <Token> is the "cookie" that gives authorization to the system, or permission to send the message. The <Token> is originally returned by the server in response to the HITISRegister message.

The following is an example of a HITIS Header::

```
<HITISMessage xmlns=" http://www.w3.org/1999/XMLSchema/datatypes " Version="1.0">
    <Header OriginalBodyRequested="false" ImmediateResponseRequired="true">
        <FromURI>http://www.pos.com/HITISInterface</FromURI>
        <ToURI>http://www.pms.com/HITISInterface</ToURI>
        <ReplyToURI>http://www.pos.com/HITISInterface</ReplyToURI>
        <MessageID>1234567890</MessageID>
        <OriginalMessageID>1234567890</OriginalMessageID>
        <TimeStamp>1999-11-10T10:23:44</TimeStamp>
        <Token>1234-567-8901</Token>
        <!--Token to be assigned in response to HITISRegister-->
    </Header>
```

6.0 HITIS REGISTER

The HITIS Register message is the first message sent to a system upon connection. It is the message that starts the initial discussion for authentication and negotiation to decide how the two systems intend to exchange messages. The HITIS Register message allows systems to define such items as the version, protocol, role, password, and language.

6.1 Language ID

The LanguageID element determines the administrative and error string language (individual messages can use different languages). The default language is English, but if another language is preferred, it would be negotiated upon registration.

6.2 Version

The Version element uses the HTML/XML convention of a string, combining both major and minor version in the element tag <Version> for example Version = 1.0. Letters may be used in combination with whole integers for interim sub-versions, e.g.: 1.1c.

6.3 Authentication

The Authentication element provides the verification of a system's identity and authorization to define the security of a connection for the exchange of messages. The Authentications (plural) element accommodates more than one security method, if such is required. The AuthenticationType attribute is a string that identifies the type of security to be applied to a session connection between parties. It is not an enumerated type, however, several types are suggested, such as x509 certificates, a Password, or a PGP digital signature, etc.

When Authentication is made the responding system returns a token. That token acts as a cookie and is used to identify the system the next time it logs on.

Depending upon the type of authentication chosen, the sub-elements<Value> and <Name>are used. If the AuthenticationType is x509 certificates, the <Value> would be a

url, and the <Name> would be the issuing authority of the certificate. If the AuthenticationType is PGP, then the <Value> would be a digital signature.

If, by contrast, two systems are on a secure local area network connection, the AuthenticationType is a Password, and the elements <Login> and <Password> would be used instead of the name/value pair.

6.4 Role

The Role element defines the permissions given to the connecting system, which may include user-defined business restrictions or privileges.

6.5 Transport Protocol

The TransportProtocol element allows two systems to define the protocol to be used to exchange messages. HITIS-compliant systems should be able to support *http*, but there is no requirement to use this transport, and the parties could agree to switch to *msnq*, *https*, SMTP, or another protocol as an alternative connection.

The originating system sends the HITIS Register message with the preferred type of protocol listed in the ProtocolType attribute. If the initial Register message is sent in *http*, and the receiving system is willing to communicate in that protocol; the responding system sends an acknowledgment in *http* and returns the HITIS Register message with the ProtocolType element and its associated <Name> and <Value> pair filled in. Once this is accomplished, the exchange of HITIS messages can begin.

If the responding system wishes to change to another transport protocol, then the information in the <TransportProtocol> element is replaced with an identification of the appropriate alternative protocol, along with the associated information in the sub-elements.

The ContactLocation element is a string used to identify a URI, a port number, or an e-mail address to which subsequent messages should be directed.

6.6 Token

The response to a HITIS Register message is another HITIS Register message returned with the Token information added in the Body. When the initial message is sent, the Token is empty. This is not necessarily a synchronous message, but the originator is not able to send any other messages until the reply is received.

Some systems may add an expiration time/date of the token, filling in the TokenExpires or the TokenCount element to indicate that the token is valid for a certain number of messages or for a certain time duration.

If a token is returned that is good for only one message, then the originating system is tasked to register again for each message, and should be efficient enough to avoid sending a message with the previous token, handling an error message, and sending a re-try. The burden lies with the client returning the token, and systems should handle this on a server request basis.

The following is an example of a HITIS Register message:

```
<HITISRegister xmlns="">
    <Header OriginalBodyRequested="false" ImmediateResponseRequired="true">
```

```
        <FromURI>http://www.pos.com/HITISInterface</FromURI>
        <ToURI>http://www.pms.com/HITISInterface</ToURI>
        <ReplyToURI>http://www.pos.com/HITISInterface</ReplyToURI>
        <MessageID>1234567890</MessageID>
        <OriginalMessageID>1234567890</OriginalMessageID>
        <TimeStamp>1999-11-10T10:23:44</TimeStamp>
        <Token/>
        <!--Token to be assigned in response to HITISRegister-->
    </Header>
    <Body>
        <LanguageID>en_US</LanguageID>
        <Version>1.0</Version>
        <Authentications>
            <Authentication AuthenticationType="Password">
                <Value/>
                <Name/>
                <Login>UserID0123</Login>
                <Password>xxxooo</Password>
            </Authentication>
            <Role>D706YX</Role>
            <TransportProtocol ProtocolType="http">
                <Property>
                    <Name/>
                    <Value/>
                </Property>
                <ContactLocation>http://www.pms.com/HITISInterface</ContactLocation>
            </TransportProtocol>
            <Token TokenType="HostAssigned"/>
            <TokenExpires>1999-11-10T24-00-000</TokenExpires>
            <TokenCount>3</TokenCount>
        </Authentications>
    </Body>
</HITISRegister>
```

6.7 HITIS UnRegister

A HITIS Unregister message notifies a system that the Token is being released along with any known information that was negotiated through the HITIS Register process. A HITIS Unregister message has no Body in the message, only the Token in the Header.

The following is an example of a HITIS UnRegister message:

```
<HITISUnRegister xmlns="" Version="1.0">
    <Header OriginalBodyRequested="false" ImmediateResponseRequired="false">
        <FromURI>http://www.pos.com/HITISInterface</FromURI>
        <ToURI>http://www.pms.com/HITISInterface</ToURI>
        <ReplyToURI>http://www.pos.com/HITISInterface</ReplyToURI>
        <MessageID>1234567890</MessageID>
        <OriginalMessageID>1234567890</OriginalMessageID>
        <TimeStamp>1999-11-10T10:23:44</TimeStamp>
        <Token>d29jtu799874903.78024.xyz</Token>
        <!--Token no longer valid upon receipt of this message-->
    </Header>
</HITISUnRegister>
```

7.0 HITIS SUBSCRIBE

The HITIS Subscribe message allows a system to subscribe to certain events, and allows the publisher of an event to handle channel definitions. The initiation of a HITIS Subscribe message is a request to receive all of the events of a type that are published. When a system registers to be a recipient of events, it subscribes to all or nothing.

Since publishing is generally done through agreements negotiated beforehand with events sent to a subscriber list, the HITIS Subscribe/ Unsubscribe messages are used to determine who has subscribed to certain events. The server keeps track of who is subscribing and what events (items) each system wishes to receive. The EventName defines what event has been subscribed to.

If a system subscribes to multiple events, there may be different servers handling the different types of events. The subscribing system sends an identification of the event it wishes to subscribe to, as it may not want to receive events for all broadcasts. Alternatively, a system may prefer to issue separate HITIS Subscribe messages in order to have the ability to subscribe to certain event types only. Subscribe messages are asynchronous and do not anticipate an immediate response.

The following is an example of a HITIS Subscribe message:

```
<HITISSubscribe xmlns="" Version="">
    <Header OriginalBodyRequested="false" ImmediateResponseRequired="false">
        <FromURI>http://www.pos.com/HITISInterface</FromURI>
        <ToURI>http://www.pms.com/HITISInterface</ToURI>
        <ReplyToURI>http://www.pos.com/HITISInterface</ReplyToURI>
        <MessageID>1234567890</MessageID>
        <OriginalMessageID>1234567890</OriginalMessageID>
        <TimeStamp>1999-11-10T10:23:44</TimeStamp>
        <Token>1234.567.8901xyzddd2i</Token>
        <!--Token assigned in response to HITISRegister-->
    </Header>
    <Body>
        <Events>
            <Event>
                <ReplyToURI/>
                <!--Address to send events when they are published-->
                <EventName/>
                <!--Identifies the type of event being subscribed to-->
                <Scope/>
                <!--Distribution channel - could include a hotel reference, a group of hotels on a campus,
                 or individual sub-systems on a property, such as the electronic lock system, PBX, etc. -->
            </Event>
        </Events>
    </Body>
</HITISSubscribe>
```

8.0 HITIS MESSAGE

A HITISMessage is a request and a response message pair. A "Request" message is the http Post that represents the function call of a HITIS operation to perform a query or to send data to another system. The "Response" message is an http Post Response that returns the out parameters of the operation, a return code integer value, and the error string(s) in the case of an error.

The content model of a HITISMessage consists of a Header and a Body. The first element of the Body is the HITISOperation which contains an OperationName= attribute that defines the context of the message. The string in the OperationName attribute determines what operation is to be performed. This is used by the parser to determine where to send the message for processing, instantiating the appropriate handler to perform the business functionality of the request.

The syntax of the (OperationName =) string is derived from the name of the operation, combined with the name of the class (if necessary to differentiate from other operations of the same name), followed by a Request or Response. The name of the DTD and XML-Schema definition files uses the OperationName string, as in the following examples:

OperationName = QueryFolioRequest returns OperationName =QueryFolioResponse, Operation Name = ReservationNotificationUpdateRequest
returns Operation Name = ReservationNotificationResponse.

Several operations from the object model were combined into one Request message if the return value was the same, resulting in the multiple input parameters in a request. Where that is the case, elements of the request message are optional, and the requestor completes only the appropriate elements needed to fulfill the request. If more than one element is filled in, the query assumes an implicit "AND" condition for the elements included.

The basic structure of a HITISMessageRequest XML document is as follows:

```
<HITISMessage xmlns=" http://www.w3.org/1999/XMLSchema/datatypes " Version="1.0">
    <Header OriginalBodyRequested="false" ImmediateResponseRequired="true">
        <FromURI>http://www.pos.com/HITISInterface</FromURI>
        <ToURI>http://www.pms.com/HITISInterface</ToURI>
        <ReplyToURI>http://www.pos.com/HITISInterface</ReplyToURI>
        <MessageID>1234567890</MessageID>
        <OriginalMessageID>1234567890</OriginalMessageID>
        <TimeStamp>1999-11-10T10:23:44</TimeStamp>
        <Token>1234-567-8901</Token>
        <!--Token to be assigned in response to HITISRegister-->
    </Header>
    <Body>
        <HITISOperation OperationName="QueryFolioRequest">
            <!--Input parameters of request message inserted here.-->
        </HITISOperation>
    </Body>
</HITISMessage>
```

The basic structure of a HITISMessage Response XML document is as follows:

```
<HITISMessage xmlns="http://www.w3.org/1999/XMLSchema/datatypes" Version="1.0">
    <Header OriginalBodyRequested="false" ImmediateResponseRequired="true">
        <FromURI>http://www.pms.com/HITISInterface</FromURI>
        <ToURI>http://www.pos.com/HITISInterface</ToURI>
        <ReplyToURI>http://www.pms.com/HITISInterface</ReplyToURI>
        <MessageID>1234567890</MessageID>
        <OriginalMessageID>1234567890</OriginalMessageID>
        <TimeStamp>1999-11-10T10:23:44</TimeStamp>
        <Token>1234-567-8901</Token>
        <!--Token to be assigned in response to HITISRegister-->
    </Header>
    <Body>
        <HITISOperation OperationName="QueryFolioResponse" HITISCode="0">
            <!--Out parameters of response message inserted here-->
        </HITISOperation>
    </Body>
    <OriginalBody>Same as Body of original request message
    </OriginalBody>
</HITISMessage>
```

The <OriginalBody> tag included in the Response Message schema is an optional element to be used if the attribute OriginalBodyRequested = true. The data type for the OriginalBody element is CDATA.

The sample response message instance above includes the HITIS Code integer value of (0), indicating Success. If the integer value is other than zero (0), the response message would include a one or more Error elements that may indicate an additional error code (hex value) and a string with the text description of the error, as in the following sample:

```
<Body>
    <HITISOperation OperationName="QueryFolioResponse" HITISCode="0x0B2F">
        <Errors>
            <Error HITISCode="0x22">BAD_FOLIO_ID</Error>
        </Errors>
    </HITISOperation>
```

```
</Body>
```

9.0 RETURN VALUES

Return values may be one of two types; 1) System errors, and 2) HITIS and Vendor-specific return values, depending upon the level of the error encountered.

The error return values are handled as follows:

9.1 System Level Errors

A system generated error value is returned by whatever mechanism the underlying system uses to return errors. e.g.: The system error handler or catch for error trapping, dependent upon how the environment handles global system errors.

9.2 HITIS Return Values

Object Values:
All HITIS operations return an integer as the final parameter. The HITIS return values are used for functional errors related to the operation. *(Exception - See **Collection Object**)*

A range of values is assigned for HITIS errors by Functionally Organized Object, using hexadecimal notation, as follows:

0 to 3FF = Generic HITIS error codes.

Generic HITIS errors exist in almost all objects, and provide a commonality across the standards. An example in the Session Authenticate operation would be Invalid_UserName, or Invalid_Password.

Then every 400 block:

400 - 7FF HITIS_POST_X (400 = HITIS_POST_BASE)

800 – BFF

C00 – FFF

…

4000 - 43FF

Up to FFFF is reserved for HITIS returns via the integer error value in the range as defined above. This allows for growth of the HITIS specifications so that other standards can be added with space available for their error definitions. *(See chart at the end of this Section)*

XML Representation:
The XML response messages return an integer to indicate success of the operation, or a HITIS or vendor-specific error condition. If a response message returns an error condition that has occurred in the processing of the Request, an integer other than zero (0) is indicated in the HITISCode attribute. Systems should independently maintain an external table for reference to the text description of the error associated with the integer.

If the response message contains data other than the HITISCode integer, those elements that contain the return data are nested inside the HITIS Operation, as in this example.

```
<Body>
  <HITISOperation OperationName="ReservationNotificationResponse" HITISCode="0">
```

```
        <ReservationID 6123456</ReservationID>
    </HITISOperation>
</Body>
```

If the HITISCode is anything other than a Zero (0) value, then an Errors element will be sent that may contain 1 to many Error elements with more detailed error information. Errors are not sent through a ReturnValues section itself in order to isolate the business contents from the infrastructure information.

```
<Body>
    <HITISOperation OperationName="ReservationNotificationResponse" HITISCode="0x2F01">
        <Errors>
            <Error HITISCode="0x20">Can't process this reservation</Error>
            <Error HITISCode="0x25">No Credit Card Information</Error>
        </Errors>
    </HITISOperation>
</Body>
```

This is HITIS Return Code 0x2407 from the published table.

0x2407 - HITIS_RESERVATION_BADHOTELID

This example assumes that the HITISCode will be interpreted as a hex value if it starts with a "0x". Otherwise, the equivalent decimal value is sent, e.g.: 9223.

9.3 Error Naming Conventions

The HITIS naming convention for error constants is as follows:

HITIS_FOO(SESSION)NAME_ERRORSPECIFICNAME

[Begins with HITIS; followed by an underscore; then the name of the object Session (The word Session does not have to be a part of it); underscore; then the error-specific name].

9.4 Range of Return Values

The range of HITIS error code returns specifically assigned to each standard are as follows: The range of vendor-specific return values would remain the same in an XML interface.

TC	Standard Name	Starting Value	Ending Value
Posting Devices	Folio Posting	0400	07FF
Posting Devices	Folio Query	0800	0BFF
Posting Devices	Room Status	0C00	0FFF
Posting Devices	Guest Messages	1000	13FF
Posting Devices	Event Notification	1400	17FF
Remote Devices	Check-In Initiation	1800	1BFF
Remote Devices	Check-Out Initiation	1C00	1FFF
Remote Devices	Security	2000	23FF
CRS	Reservation Synchronization	2400	27FF
CRS	Profile Synchronization	2800	2BFF

CRS	Guest Stay Information	2C00	2FFF
CRS	Statistics	3000	33FF
CRS	Agent Commissions	3400	37FF
CRS	Availability, Rate & Inventory	3800	3BFF
CRS	Availability Query & Booking Request	3C00	3FFF
		4000	43FF
			etc.

9.5 Vendor-Specific Return Values

A vendor-specific return value should return an integer as the final parameter. Any error return integer after 10000 hex is vendor-defined. It is recommended that the description for each operation should list the possible errors that could be returned.

Error	Value	Description
VENDORSPECIFIC_START		(value in excess of 10000 hex)
VENDORSPECIFIC_END		(value not limited by HITIS)

The range of vendor-specific return values would remain the same in an XML interface.

10.0 COLLECTION OBJECT (Generalized Class / Template)

Object Values:

A Collection object provides a consistent way to contain multiple like objects within another object. An example is the collection of guests associated with a reservation, or a collection of rooms associated with a room stay. This type of object is used throughout the HITIS interfaces.

Since the Collection class is use is often repeated within an object, it is advantageous to define a Generalized class that acts as a template for code generation. The generalized class is assigned a specific name when the instance of that collection is instantiated, which is named with the plural form of the singular object that it collects.

10.1 Attributes

6.1.1 Count : Unsigned Integer

A read only attribute that returns the number of items in the collection. A Count is a value from 1 to N. (not zero to N). When an object is added to a collection, it adds it to the end of the index of the collection, and changes the Count of the total collection.

10.2 Operations

6.1.2 Item (Index : Unsigned Integer) : Object

The Item operation is a read only operation that retrieves a specific item from the collection, and implies that a collection can be ordered. All collections are 1 based (not zero-based) for the index.

6.2.2 Add (NewItem : Object)

The Add operation inserts a new item into the collection. It defines the operation necessary for the local interface to add an object to the "empty bucket" of the collection. Each collection only accepts objects of a certain type; that is, each specific collection class derived from this parent template will have its own Add operation that only accepts the appropriate type of object. (Similarly, each will have its own Item operation that only returns the appropriate type of object.) This provides a consistent way to fill an empty collection.

When collections are created by the client system, they are read/write, but at the point at which they are passed across the wire the data is considered a snapshot of the client system and they become read-only. Only the Count attribute and Item operation of the Collection class are sent to the receiving system, as the receiver would not be able to Add to the collection. Collection objects are an exception to the normal convention that every operation returns an integer. There is no Remove operation in the collection class.

10.3 Collection Object (Instantiated Class)

The operations of the Collection object are repeated in the instantiated collection subclass with the name of the objects to be collected replacing the generic "Object". The named subclass inherits the attributes and operations of the Collection object. When the Item operation is repeated in the named collection class, it constrains or narrows the object to be returned to the specific object of that collection, and the specific name overrides the generic Object.

For example, the Profiles collection would contain the following operations:

Item (Index : UInt) : Profile
Add (New Item : Profile)

6.3.1 Item (Index :Unsigned Integer) : *[ObjectName]*

The Item operation is a read /write operation that retrieves a specific item from the collection, and returns the object. The Index operation implies that a collection can be ordered. All collections are 1 based (not zero-based) for the index. The Index argument can remove the object if the value is NULL at the index.

6.3.2 Add (NewItem : *[ObjectName])*

The Add operation is a read/write operation that inserts a new item to the collection, and defines the operation necessary for the local interface to add the specific object to be collected into the collection.

The results of the generic Collection class operations and specific named class operations are the same, but the specifications are written with the operations in both classes so that the option of binding can be done either at compile time or late binding at runtime.

The initialized, named Collection class assumes that the collection is accessed one at a time, and that it is accessible by index. The named Collection class is a snapshot of the data collected. It is

read/write when it is in the local environment, it becomes read-only when the Collection is passed in the interface.

XML Representation:
A collection class does not exist, per se, in the XML messaging model; that is, no separate class containing operations or attributes is defined in UML. However, the consistent naming convention is used throughout HITIS interfaces for containing multiple like elements within a single element. The collection class is the plural form of the singular object that it collects. e.g.: FolioDetails **:** FolioDetail.

The content model of the plural element will have zero or more sub-elements of the singular name.

11.0 ATTRIBUTE (PROPERTY) NAMING

Object Values:
Note that the term "entity" is used in this section instead of "object" (hopefully to reduce confusion), as some attributes may refer to something that has no corresponding HITIS object. For example the GuestMembership interface class in the Profile Synchronization object has an "AccountID" attribute that is the guest's account number (alphanumeric) in the particular frequent guest program. This attribute performs the task of uniquely identifying something (the account), but that something is not a HITIS object.

11.1 ID

Any alphanumeric (string) attribute that uniquely identifies some entity should be named "xID", where "x" is the entity being referenced. For example, a room "number" in a hotel would properly be termed "RoomID" (since not all room numbers are strictly numeric).

11.2 Number

Any numeric (Integer or Ulong) attribute that uniquely identifies some entity should be named "xNumber", where "x" is the entity being referenced. For example, the attribute correlating RoomStays to Guests in the RoomStay object is called GuestNumbers.

11.3 Code

Any alphanumeric (string) attribute that is a mnemonic abbreviation of some sort, or a value that is translated between systems should be named "xCode", where "x" is the entity being referenced. For example, the state and country portions of a mailing address are properly termed "StateCode" and "CountryCode", with the understanding that the commonly used abbreviations are what they contain. For example, a rate plan reference (which must be translated between systems, and refers to a rate table) is named "RatePlanCode".

11.4 Type

Any numeric attribute whose value is one of an enumerated list of HITIS defined values is named "xType", where "x" is the entity being referenced. For any "type" attribute, the interface specification defines a UInt-derived datatype that limits the allowable values. The sequence of allowable values is 1-based, with zero (0) being reserved for the value of NULL, Does Not Apply, etc.

It is likely that HITIS will continue to add types to an enumerated list, and that vendors will also have types unique to their own implementation. The HITIS standards reserve the numbers

through 999 for HITIS-defined types, with integers beginning with 1000 designated for vendor-specific types.

11.5 Collections

Any object that is a collection of like objects is named the plural of the object in the collection. For example, the “Guests” object in Reservation Notification is a collection of Guest objects.

XML Representation:
The HITIS messaging model in UML has retained the names of the class attributes and they were mapped to element names in XML. All 'xType' attributes were mapped to attributes` instead of elements with the data type of **enumeration**. The constraints of the enumeration are the HITIS-defined 'xType' values.

ATTRIBUTE TYPES - XML STRINGS
The World Wide Web Consortium (W3C) also defines enumerated types (notations and enumerations) and a set of tokenized types (for example, ID, IDREF, and NMTOKEN).

entity	Represents the XML ENTITY type.
entities	Represents the XML ENTITIES type.
enumeration	Represents an enumerated type (supported on attributes only).
id	Represents the XML ID type.
idref	Represents the XML IDREF type.
idrefs	Represents the XML IDREFS type.
nmtoken	Represents the XML NMTOKEN type.
nmtokens	Represents the XML NMTOKENS type.
notation	Represents a NOTATION type.
string	Represents a string type.

11.6 Built-in vs. user-generated datatypes

Built-in datatypes are those which are entirely defined by the W3C specification, and may be either primitive or generated; **User-generated** datatypes are those generated datatypes that are defined by individual schema such as HITIS by giving values to constraining facets.

Conceptually there is no difference between the built-in generated datatypes included in the W3C specification and user-generated datatypes created by individual schema designs. The built-in generated datatypes are those that are common to many schemas and are designed so that systems other than the XML Schema Definition Language may access them.

12.0 OPERATION (METHOD) NAMING

Object Values:
For operations defined in the HITIS standards, the parameters (arguments, message signatures) are not optional. However, a value is optional and can be left open with a placeholder. For example, an empty string or null value, etc.

12.1 Create

Any operation that creates (i.e., instantiates) a new local (usually empty) container and returns it to the client application should be named “CreateX”, where "x" is the name of the type of

object being returned. Objects that contain operations have explicit Create functions provided in their containing object. All objects that are exclusively data collectors are directly creatable by client applications. Create operations are local operations. Although they may exist in the underlying architecture, they are publicly exposed in the HITIS standards to provide a clearer understanding of the hierarchy of the object design and consistency for mapping.

12.2 Update

Any operation named "Update" is one that commits a change in the data of the object and sends the new or modified object to the destination system (server). A single update is an operation of the object being updated, not an operation of the Session object. A batch update is an operation of the containing object.

12.3 Query

Any operation named "Query" is one that requests information from the system and returns an integer that indicates success or a HITIS / vendor-specific error.

XML Representation:
The HITIS messaging model in UML retained the names of the class operations when they were mapped to message names in XML. The Update operations have been named from the class they originated from, e.g.: a CommissionEventUpdateRequest message, which returns a CommissionEventUpdateResponse message. The Query operations became Query message pairs, e.g.: QueryProfileRequest, which returns a QueryProfileResponse. This same naming convention was used with "Get" operations, e.g.: GetProfileRequest which returns a GetProfileResponse.

The Create operation is no longer needed in the XML mapping as it was used to instantiate an object on a remote system.

13.0 NAMESPACES

NOTE* - The revised, February 25, 2000, W3C specification, Section 3.1 defines **Namespace considerations** as follows:

3.1 Namespace considerations

The built-in datatypes defined by this specification are designed so that systems other than the XML Schema Definition Language may use them. To facilitate such usage the built-in datatypes in this specification have the namespace URI:

- **http://www.w3.org/1999/XMLSchema/datatypes**

This applies to both built-in primitive and built-in derived datatypes.
Each user-generated datatype is also associated with a unique namespace. However, user-generated datatypes do not come from the XML Datatype Language namespace; rather, they come from the namespace of the schema in which they are defined.
Each user-derived datatype must be defined in terms of another datatype in one of two ways: 1) by assigning constraining facets which serve to restrict the value space of the user-derived datatype to a subset of the base type: 2) by creating a list datatype whose value space consists of finite-length sequences of values of the base type.

Note that associating a namespace with a user-generated datatype is not a general purpose extensibility mechanism and does not apply to primitive datatypes. Schema processors may not understand a user-generated datatype simply by defining the datatype and associating a unique

namespace with them. However, all schema processors are required to be able to validate datatypes defined by subsetting the value space of a "built-in" datatype.

The unique HITIS datatypes defined in this document are individually defined in each HITIS DTD and XML-schema definition file, instead of referring to a namespace to define a HITIS metadata type.

14.0 REFERENCES

ISO 4217:1995 - Codes for the representation of currencies and funds

Maintenance Agency
c/o British Standards Institution
389 Chiswick High Road
London W4 4AL
United Kingdom
Telephone: + 44 181 996 9000
Telefax: + 44 181 996 7400
Telex: 82 57 77 bsi mk g

ISO 639:1988 - Code for the representation of names of languages

Registration Authority
International Information Centre for Terminology
(INFOTERM)
Simmeringer Hauptstrasse 24
A-1110 Wien
Austria
Telephone: + 43 1 740 4 0 441
Telefax: + 43 1 740 40 740

ISO 3166-1:1997 - Codes for the representation of names of countries and their subdivisions
Part 1: Country codes

ISO 3166 Maintenance Agency
c/o DIN Deutsches Institut für Normung
Burggrafenstrasse 6
D-10772 Berlin
Germany
Telephone: + 49 30 2601 2791
Telefax: + 49 30 2601 1231
e-mail: lechner@nabd.din.de

XML Schema Part 2: Datatypes
W3C Working Draft 25 February 2000

W3C Office at RAL
Rutherford Appleton Laboratory
Chilton, Didcot
Oxfordshire OX11 0QX
United Kingdom
Telephone: +44 1235 446822
Telefax: +44 1235 445385
e-mail: w3c-ral@inf.rl.ac.uk
Online: http://www.w3.org/TR/xmlschema-2

HITIS Interface Standard

Glossary of Terminology

Glossary of Terminology

Abstract Class	A specialized class used solely for subtyping. It defines a common set of behaviors to be inherited by its subtypes. It has no instances. (Synonymous with Virtual Class in C++).
Account	A record of transactions entered as credits and debits under a particular name, such as a hotel guest.
API	Application Programming Interface, through which an application communicates with some underlying resource on a computer system. The underlying resource can be either hardware or software.
Application	One or more programs or associated data running on a computer system to facilitate user requirements. In the hospitality environment, these would include Property Management System applications and Central Reservation System applications.
Architecture	A high-level description of the organization of functional responsibilities within a system. Many different levels of architectures are involved in developing software systems, from physical hardware architecture through the logical architecture of an application framework.
Association	Meaningful links between objects. A person associated with a company creates the concept of employment.
Attribute	An identifiable association between an object and a value. An attribute A is made visible to clients as a pair of operations: get_A and set_A. Read only attributes only generate a get operation. [OMG] A characteristic or property of an object. Usually implemented as a simple data member or as an association with another object or group of objects.
Bandwidth	The theoretical speed at which data can be transmitted over some media. The actual throughput may be lower due to factors such as network contentions.
Base Class	A class that has one or more derived classes that inherits its attributes and methods. (Synonymous with Superclass).
Behavior	The behavior of a request is the observable effects of performing the requested service (including its results).
Binding	The selection of the method to perform a requested service and of the data to be accessed by that method.
Blind Post	Post without a query.

Browser	A software application (utilizing HTML) that enables a user to connect to the Internet to gain access to the vast stores of information on the World Wide Web. Whether a user searches for information or having it delivered to their computer, it enables the user to make it easy to browse the Internet. There are also browsers for using the other languages within SGML.
CDATA	See Non-Parsed Character Data.
Class	An implementation that can be instantiated to create multiple objects with the same behavior. An object is an instance of a class. Types classify objects according to a common interface; classes classify objects according to a common implementation.
Class Attribute	A characteristic or property that is the same for all instances of a class. This information is usually stored in the class type definition.
Class Hierarchy	Embodies the inheritance relationships between classes.
Class Inheritance	The construction of a class by incremental modification of other classes.
Class Member	A method or an attribute of a class.
Class Method	A class method defines the behavior of the class. Such a method performs tasks that cannot or should not be done at the instance level, such as providing access to class attributes or tracking class usage metrics.
Class Object	An object that serves as a class. A class object serves as a factory.
Client	An intelligent workstation, typically a Personal Computer, running front-end applications in a client/server environment. This object may request a service from a server object in a client/server relationship. The code or process that invokes an operation on an object
Client/Server	A relationship between a client that requests services and servers that provide services. This relationship is paralleled in an Object Oriented (OO) environment by message senders and receivers.
Code	Instructions that guide the tasks performed via a computer program.
Collaboration	Two or more objects that participate in a client/server relationship in order to provide a service.
COM	Component Object Model. The object model, which defines the relationships between the various components that make up an interface. A binary specification that enables software suppliers to package functions into resusable software components in a fashion similar to the integrated circuit.
Component	A conceptual notion. A reusable piece of software in binary form that can

	be plugged into other components from other vendors with relatively little effort. Classes, systems or subsystems that can be designed as reusable pieces. These pieces can then be assembled to create various new applications.
Configuration Management	The discipline of identifying a system and its component parts at discrete points in time. Monitoring throughout versions and revisions enables CM to systematically control changes to maintain integrity and traceability of the system throughout a product's lifecycle. This includes hardware, environment, code, documents and objects.
Container Class	A class designed to hold and manipulate a collection of objects.
Data Model	A collection of entities, operators and consistency rules.
Data Type	A categorization of values, operations and arguments, typically covering both behavior and representation (e.g., the traditional non-OO programming language notion of type).
DCD	The Document Content Description a structural schema facility, for specifying rules covering the structure and content of XML documents. DCD is intended to define document constraints in XML syntax; these constraints may be used in the same fashion as traditional XML DTDs. DCD also provides additional properties, such as basic datatypes.
Default	Value used if nothing is chosen.
Derived Class	The class created through inheritance. A derived class inherits the methods and attributes of its superclass(es) and usually adds its own to distinguish its capabilities or services.
DOM	The Document Object Model provides an abstract API for constructing, accessing, and manipulating XML and HTML documents. A ‘binding’ of the DOM to a particular programming language provides a concrete API.
Domain	A formal boundary that defines a particular subject or area of interest.
DTD	A Document Type Definition is a file (or several files to be used together), written in XML, which contains a formal definition of a particular type of document. It sets out what names can be used for element types, where they may occur, and how they all fit together.
Dynamic Link Library	A dynamically loaded run-time library.
EDI	Electronic Data Interchange has been used in e-commerce for many years to exchange documents between commercial partners to a transaction. It has required special proprietary software, but there are now moves to enable EDI data to travel inside XML.

Element	Any container in an XML document. Elements may contain other Elements, processing instructions, and DTD characters.
Element Tree	This is a collection of Elements contained in an XML document.
EPOS	Electronic Point Of Sale.
Encapsulation	The technique used to hide the implementation details of an object. The services provided by an object are defined and accessible as stated in the object contract.
Encryption	A process for scrambling access codes to computer programs to prevent illicit entry into and control of the system.
Enterprise Modeling	A technique for modeling an entire business enterprise from the business manager's point of view. An enterprise model is composed of the objects, events and business rules that describe the enterprise. Separate but related business systems can be built from this model to enhance the efficiency and consistency of the operation of the enterprise.
Error String	Test of an error message indicating the nature of the error.
Event	Something that takes place asynchronously, such as the pressing of a computer key or clicking of a mouse.
Event	A significant change in the environment or the state of an object that is of interest to another object or system.
Failure	When a program or piece of software fails to reach some desired result. In some cases, this will generate an error string.
Firewall	A computer system that sits between a host system and user system, and protects the host from unauthorized access to protected data and applications.
FMS	Food Management System. The software that focuses on the food production cycle from ordering to production to accounting.
Folio	A ledger page detailing charges to an account.
FOO	Functionally Organized Object. The sets of data and message handling between two systems that need to communicate. An interface could be one FOO or several. Likewise, a FOO could be used in several interfaces (e.g., Folio Posting).
Formal Parameter	A named local object used as an argument to an operation. The value of the object (actual parameter) is assigned by the client who runs the method.
Generalization	The inverse of the specialization relation.

Graphical User Interface (GUI)	A graphical way of displaying data, as opposed to text-based programs, that users can access through simple point and click mouse routines. Any interface that communicates with the user, primarily through graphical icons.
HTML	This is the HyperText Markup Lanaguage, a specific application of the SGML. A predefined markup language like HTML defines a way to describe information in one specific class of documents. It defines a simple, fixed type of document with markup designed for a common class of office or technical report, with headings, paragraphs, lists, illustrations, etc, and some provision for hypertext and multimedia.
Implementation	A definition that provides the information needed to create an object and allow the object to participate in providing an appropriate set of services. An implementation typically includes a description of the data structure used to represent the core state associated with an object, as well as definitions of the methods that access that data structure. It will also typically include information about the intended interface of the object.
Import	Creating an object based on a description of an object transmitted from an external entity.
Inheritance	The construction of a definition by incremental modification of other definitions. (See also Implementation Inheritance)
Initialization	Setting the initial attribute values of a new object.
Input	Data going into a computer system.
Instance	An object created by instantiating a class. An object is an instance of a class.
Instance Variable	A variable that contains a value specific to an object instance.
Instantiation	Object creation.
Interface	A description of a set of possible uses of an object. Specifically, an interface describes a set of potential requests in which an object can meaningfully participate.
Interface Description Language (IDL)	Interface description language describing the interfaces of an object. From the IDL, a compiler can generate header files for programming languages so that applications can use that interface, create proxy and stub objects to provide for remote procedures calls, and create the output necessary to enable RPC (Remote Procedure Calls) across a network. When used in conjunction with an ORB, IDL statements describe the properties and operations of an object.
Interface Type	A type that is satisfied by any object (literally, by any value that identifies

	an object) that satisfies a particular interface.
ISO 9000 Certification/Standards	The International Organization for Standardization (ISO) issues the ISO-9000 guidelines for the selection and use of the series of standards on quality systems.
Java	Programming language designed to work in a distributed environment.
Launch	To begin execution of a software application and/or program.
Legacy System	An older computer system based on previous generation hardware and software. Typically, these systems are mainframe-based or mini-computer systems.
Link	Relation between two objects (a concept).
Literal	A value that identifies an entity that is not an object.
Login	Procedure used to authenticate a user and to enable the user to access a network.
Macintosh	A personal computer designed and marketed by Apple Computer (which pioneered the GUI (Graphical User Interface)-based operating system).
Mapping	A rule or process, the O-O equivalent of a mathematical function. Given an object of one set, a mapping applies its associative rules to return another set of objects. Member Function (See Method)
Message	The mechanism by which objects communicate. A message is sent by a client object to request the service provided by the server object.
Meta-Model	A model that defines other models.
Method	Code that can be executed to perform a requested service. Methods associated with an object are structured into one or more programs.
Non-Parsed Character Data	(CDATA). The content or attribute value of an Element that consists of text that should be processed by the parser.
Object	A combination of a state and a set of methods that explicitly embodies an abstraction characterized by the behavior or relevant requests. An object is an instance of a class. An object models a real world entity and is implemented as a computational entity that encapsulates state and operations (internally implemented as data and methods) and responds to requests for services. [OMG] An object is a self-contained software package consisting of its own private information (data), its own private procedures (private methods), which manipulate the object's private data, and a public interface (public methods) for communicating with other objects.
Object Creation	An event that causes an object to exist that is distinct from any other object.

Object Interface	A description of a set of possible uses of an object. Specifically, an interface describes a set of potential requests in which an object can meaningfully participate as a parameter. It is the union of the object's type interfaces.
Object Model	Conceptual model for describing how components interact on a computer system. (See Component Object Model.)
Object Name	A value that identifies an object. (See Handle)
Object Reference	A value that precisely identifies an object. Object references are never reused to identify another object.
Object State	The current information about an object that determines its behavior.
Object Wrapper	The result of encapsulating a set of services provided by a non O-O application or program interface in order to treat the encapsulated application or interface as an object.
Object-Oriented	Any language, tool or method that focuses on modeling real-world systems using the three major components of objects – encapsulation, inheritance and polymorphism.
Object-Oriented Design	The process of developing an implementation specification that incorporates the use of classes and objects. It encourages modeling the real world environment in terms of its entities and their interactions.
Operation	A service that can be requested. An operation has an associated signature, which may restrict which actual parameters are possible in a meaningful request.
Operation name	A name used in a request to identify an operation.
Output	Data produced by a computer system. It may be contained in several forms (e.g., report, file, etc.).
Paradigm	A broad framework for thinking about and perceiving reality. A theoretical, philosophical model composed of identifiable theories, laws and generalizations used in defining and solving problems.
Parameter	A value passed to a program or function via an operation/method.
Parameterized Class	A class that allows users to declare member functions and data members of "Some Type," which can be used as a template for declaring specialized subclasses that supply the "Missing" types.
Parsed Character Data	(#PCDATA). Element content that consists of text that the parser processes aside from all other markup text and non-parsed data.
Parser	A parser is a software application that processes languages (using specific

	rules) and creates machine-readable code that may be executed by another application. In XML, it also determines whether the XML code is 'valid' or 'well-formed', and passes the set of correct XML code to a downstream application.
Plain Text	Usually refers to unformatted text on a computer system supporting a GUI. Can also refer to similar text printed out on paper, usually of a fixed pitch and print size.
PMS	A Property Management System is a set of software application(s) used to help managing a facility (e.g., hotel property).
Polymorphism	The concept that two or more types of objects can respond to the same request in different ways.
Post	To apply a value or transaction to a particular account.
Property	A conceptual notion. An attribute, the value of which can be changed.
Protocol	Mutually agreed upon rules by which two systems communicate with one another.
Public	A scoping mechanism used to make member access available to other objects.
Query	A request for information contained in a database.
Relation	An object type that associates two or more object types. A relation is how associations are formed between two or more objects.
Repository	Usually a central location used to store and organize software components and related definitions, rules, etc.
Request	An event consisting of an operation and zero or more actual parameters. A client issues a request to cause a service to be performed. Also associated with a request are the results that can be returned to the client. A message can be used to implement (carry) a request and/ or a result.
Requirements	A document describing what a software system does from a user's point of view. This document is input into the object-oriented analysis process, where it will be transformed into a much more precise description.
Result	The information returned to the client, which can include values as well as status information, indicating that exceptional conditions were raised in attempting to perform the requested service.
Reuse	Reuse is the process of locating, understanding and incorporating existing knowledge, design and components into a new system. Reuse should occur at all levels of system development analysis, design, implementation, testing, documentation and user training.

Role	A sequence of activities performed by an agent.
Rule	Rules exist in two types: constraints and generic functions.
Scalability	The ability of a system to grow without sacrificing performance.
Server	A shared computer in a networked client-server environment that can provide application services to many "client" users simultaneously.
Service	A computation that can be performed in response to a request.
SGML	This is the Standard Generalized Markup Language, the international standard for defining descriptions of the structure and content of different types of electronic document. SGML is the `mother tongue', used for describing thousands of different document types in many fields of human activity.
Signature	Defines the parameters of a given operation including their number order, data types and passing mode; the results, if any; and the possible outcomes (normal vs. exceptional) that might occur.
State	The information about the history of previous requests needed to determine the behavior of future requests.
Strongly Typing	A language characteristic that requires an explicit type declaration for every value or expression. Strong typing makes static binding feasible.
Stub (Code)	A local procedure corresponding to a single operation that invokes that operation when called.
Superclass	A class that provides its methods and attributes to another class derived from it via inheritance.
System	The computer hardware-software and network configuration that enables a user to perform some tasks.
Target Object	An object that receives a request. (Synonymous with Server Object)
Transaction	An event, as in a purchase, initiated through a POS system (generally involving the purchase of goods and/or services).
Trigger Rule	A cause-and-effect relationship. When a certain event type occurs, a specific operation will be performed.
Type	A predicate (Boolean function) defined over values that can be used in a signature to restrict a possible parameter or characterize a possible result. Types classify objects according to a common interface; classes classify objects according to a common implementation.

Unicode	The Unicode Worldwide Character Standard is a character coding system designed to support the interchange, processing, and display of written texts of the diverse languages of the modern world. Unicode also supports classical and historical texts of many written languages. In its current version (2.0), the Unicode standard contains 38,885 distinct coded characters derived from 25 supported scripts. These characters cover the principal written languages of the Americas, Europe, the Middle East, India, Asia and Pacifica.
UNIX	A multi-tasking, 32-bit operating system with roots in Bell Labs. Today, various flavors of UNIX run on a variety of hardware platforms.
Use Case/ Scenario	A description of the sequence of actions that occurs when a user participates in a dialogue with a system. It describes the behavior that is invoked by a system function.
UserID	Information entered into a system (i.e., a logon) needed to identify and begin a user session.
Valid	An XML document is said to be valid if its structure and element content is formally declared in a DTD.
VBA	Visual Basic for Applications.
Verify	To check that the result of an operation is what was expected.
Visual Basic	A graphical design tool used to generate software applications. Can also refer to the programming language used by that tool.
Well formed	An XML document is said to be well formed if its structure and element content does not require a DTD.
Windows	A multi-tasking, 32-bit, GUI-based operating system developed by Microsoft in the early 1990s.
XML	The eXtensible Markup Language (XML) is a data format for structured document interchange on the Web (Internet). It is extensible because it is not a single, fixed format, predefined markup language like HTML. XML is a metalanguage – a language for describing other languages, which lets you design your own markup. It is a subset language of SGML.
XML Namespaces	XML namespaces provide a simple method for qualifying element and attribute names used in Extensible Markup Language documents by associating them with namespaces identified by URI references.