OTM-DE opentravel2.0 model designer

opentravel setting the standard

About OpenTravel:

The OpenTravel Alliance provides a community where companies in the electronic distribution supply chain work together to create an accepted structure for electronic messages, enabling suppliers and distributors to speak the same interoperability language, trading partner to trading partner. Tens of thousands of the OpenTravel message structures are in use, carrying tens of millions of messages between trading partners every day.

Members do the work of identifying what messages are needed, prioritize the work and collaborate to create the messages. Members who are looking for more information on related project team work or who wish to access the OTM repository can send inquiries to architecture@opentravel.org.

Note: This document supports implementers using the OTM-DE Model Builder in the creation and sharing of models that automatically generate xml schema. The ability to access and extend the OpenTravel Model is available only to OpenTravel members. For more information please contact us at membership@opentravel.org.

OTM-DE Reference Language Specification

Document Purpose:

The purpose of this document is to define the format, constructs, and semantic business rules of an OpenTravel Model.

Table of Contents

Table of Contents

1 Introduction

The OpenTravel 2.0 project originally began as an attempt to define a better and more technology-friendly style guide for XML schemas. Soon after its inception, however, it became clear that that the resulting OTA2.0 style guide would be difficult, if not impossible, for an individual to follow while authoring hand-crafted schemas. At that point, the effort shifted from documenting XSD authoring guidelines to the definition of a formal information modeling language.

This language, later dubbed the OpenTravel Modeling language (OTM), could be used to define the data models required for travel industry data interchange. Those models could then be compiled into XML schemas using the technical standards that were originally to be specified in the style guide manual. This approach eliminates the need for extensive XML knowledge during the model design process. Once complete, the designer can generate valid and technology binding-friendly schemas that can immediately be utilized by application development teams.

The purpose of this document is to define the format, constructs, and semantic business rules of an OpenTravel Model. The details of XML schema generation will not be addressed here, but may be found in the OTA2.0 Schema Compiler Specification document.

2 Change History

Revision	Author(s)	Summary of Changes
1.0	S. Livezey	Initial Draft

3 Referenced Documents

A number of the chapters and sections of this document reference the following documents:

1. OTA2.0 Library Schema, version 1.4.6 (included in OTM-DE Model Designer download)
2. OTA2.0 Project Schema, version 1.0.0 (included in OTM-DE Model Designer download)

4 Documentation Conventions and Terminology

This section introduces the typography and highlighting used in this document to present various types of technical material.

Normative Terms

Within the prose of this document, the following terms denote normative specifications and are defined as follows:

MAY	Models, documents, and processors are permitted to function in the stated manner but need not behave exactly as described.
SHOULD	It is recommended that models, documents, and processors function in the stated manner, but there may be valid reasons for them not to; it is important that the full implications be understood before adopting behavior at variance with the recommendation.
MUST	Models, documents, and processors are required to behave as described in order to be in compliance with this specification.
MUST NOT	Models, documents, and processors are forbidden to behave as described in order to be in compliance with this specification

Semantic Validation Terminology

When describing the semantic validation rules of an OTM model, each of the possible evaluation criteria is qualified with of the following severity levels:

ERROR	Occurs when a processor detects the violation of a behavior specified using the normative terms of "MUST" or "MUST NOT". Under this condition, follow-on processing cannot proceed until all such non-compliances have been corrected.
WARNING	Occurs when a processor detects the violation of a behavior specified using the normative term of "SHOULD". Under this condition, follow-on processing may proceed without the correction of any such non-compliance.

Property Lists

Many sections of this document provide tables of information that describe the content or properties for some aspect of the OTM modeling language. Unless otherwise stated, the information provided in these lists should be considered normative.

The following table demonstrates the format used to present property lists within this document:

Property Name	Property Type	Description
Property 1	String	Property 1 description…
Property 2	Integer	Property 2 description…
Property 3	IDREF	Property 3 description…

Unless otherwise specified, the 'Property Type' column of these tables should be assumed to reference a standard XML data type or some other type defined in the OTA2.0 Library Schema document (see Appendix A for details).

Descriptive Content and References

In many cases non-normative content is provided to more effectively describe or demonstrate points from the normative sections of the document. The principle types of content provided in this manner include *examples* and *schema excerpts*.

The following table demonstrates the format used to present example content within this document:

Example: Example Title

```
<example name="bar">
  <type>foo</type>
</example>
```

The following table demonstrates the format used to present schema excerpts within this document:

Schema Excerpt: Excerpt Title [schema-filename.xsd]

```
<xs:complexType name="example">
  <xs:sequence>
    <xs:element ref="type"/>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string"/>
</xs:complexType>
```

5 Open Travel Model

An Open Travel model is composed of one or more libraries, each of which contains a number of vocabulary terms. Each model is considered to be the "universe" of all libraries that have been loaded at any given time for the purpose of editing and/or compilation into derived formats such as XML schemas.

6 Open Travel Libraries

Open travel models can contain any number of three different types of libraries, each of which are described in further detail in this section:

1. User-Defined Libraries
2. Legacy XML Schemas
3. Built-In Libraries

6.1 Common Library Characteristics

Each type of library contains a collection of entity definitions or terms that are considered to be global definitions within an Open Travel model. In addition to its list of terms, all are composed of the following properties or characteristics:

Property Name	Property Type[1]	Description

[1] For a detailed listing of property types, see the listing of simple types in the schema definition of the Library XML format provided in the OTA2.0 Library Schema (see the Referenced Documents section for more information).

Property Name	Property Type[1]	Description
Library URL	URL	Denotes the URL location from which the library's content was loaded. This value is not explicitly specified in the content of the library; instead, it is identified by the URL location of the content itself. Relative URI's that are specified within a library are always treated as a relative path from this URL.
Name	Name_File	The name of the library
Namespace	URI	The target namespace of the library; all terms defined in the library are assigned to this namespace
Prefix	String	Short identifier used as an alias for the library's namespace when its terms are referenced by the content of other libraries
Includes	URI	Relative URI reference(s) to dependent libraries assigned to the same namespace (see Section 6.1.1)
Imports	Library Import	References to dependent libraries assigned to a different namespace (see Section 6.1.2)

In many cases, the terms of a library will refer to terms that are defined in other libraries. In these situations, the referring library MUST define a reference to its dependencies using import and/or include directives. During the loading process, all of the specified import/include dependencies MUST be resolved in order for the resulting model to be considered valid.

6.1.1 Library Includes

A library include is a relative URI reference to the library URL location of a dependent library that is assigned to the same namespace as the referencing library. The exact format of the library include MAY change depending upon the type of library, but the function of an include is identical, regardless of its format.

6.1.2 Library Imports

A library import specifies a reference to a dependent library that is assigned to a different namespace than the referencing library. While the format of an import varies by library type, each import provides the following information:

Property Name	Property Type	Description
Namespace	URI	The namespace of the library or libraries to be imported
Prefix	String	The alias to be used within the importing library when referencing terms that are assigned to the imported namespace; import prefixes MUST be unique within the importing library
File Hint	String	Space-separated list of relative URI's for libraries being imported

6.2 User-Defined Libraries (OTM)

User-defined libraries, commonly known as OTM files because of their '.otm' file extension, serve as containers for vocabulary terms of the OTM modeling language. In addition to the common properties defined in section 6.1, user-defined libraries provide the following properties:

Property Name	Property Type	Description
Version Scheme	String	An identifier that indicates the scheme used for version numbering, including how version identifiers are encoded into a library's namespace URI. At the time of this document's publication, the only supported version scheme identifier is "OTA2".
Status	Library_Status	Indicates the maturity level and editability of a user-defined library. Valid values are "DRAFT" and "FINAL". The contents of DRAFT libraries MAY be modified; libraries in FINAL status are considered locked and MUST NOT be changed.
CRC Value	Long Integer	If the status of a library is FINAL, this property will contain a long integer value that is the calculated CRC for the library's content. This is used to determine if the library's content has by manually modified after assigning it to FINAL status.
Comments	String	A textual description or other remarks related to the user-defined library

The following example XML snippet contains an excerpt from a user-defined library:

Example: Sample User-Defined (OTM) Library

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xmlns="http://www.OpenTravel.org/ns/OTA2/LibraryModel_v01_04"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <VersionScheme>OTA2</VersionScheme>
    <Status>Final</Status>
    <CrcValue>0123456789</CrcValue>
    <Namespace>http://www.OpenTravel.org/ns1/v1</Namespace>
    <Prefix>ns1</Prefix>
    <Name>Foo_Library</Name>
    <Includes>Bar_Library_1_0_0.otm</Includes>
    <Import prefix="fcom" fileHints="Foo_Common_1_0_0.otm"
        namespace="http://www.OpenTravel.org/ns1/common/v1"/>
    <Import prefix="xs" namespace="http://www.w3.org/2001/XMLSchema"/>

    <!-- List of library terms... -->

</Library>
```

User defined libraries MAY contain the following terms (see Section 9 for details):

- Simple Types
- Closed Enumerations
- Open Enumerations
- Values With Attributes
- Core Objects
- Business Objects

- Service

Multiple instances of each type of term are allowed, except for services. User-defined libraries MUST NOT contain more than one service definition per library file.

6.3 Legacy Schemas (XSD)

In some cases, it is necessary for an OTM user-defined library to reference types and/or elements that are contained in a non-OTM schema. For this reason the OTM modeling language supports references to certain entities defined in legacy XML schema documents. Legacy schemas MUST conform to the Schema for Schemas format as described in the W3C Recommendations for XML Schema, Part 1 (http://www.w3.org/TR/xmlschema-1/#normative-schemaSchema).

User-defined libraries that import or include legacy schemas MUST NOT reference legacy schema terms that are not among the following (see Section 9.10 for details):

1. XSD Simple Types
2. XSD Complex Types
3. XSD Global Elements

6.4 Built-In Libraries

Built-in libraries are typically bundled with the processors used to parse and load the contents of an OpenTravel model. As pre-bundled content, these built-in libraries are automatically present in any model, and MAY take the form of either user-defined libraries (.otm) or legacy schemas (.xsd).

Because built-in libraries are automatically loaded into all Open Travel models, there is no need for user-defined libraries to provide a specific include directive for the built-in library (assuming the user-defined library is assigned to the same namespace as the built-in). When importing a built-in library, it is only necessary to specify the namespace of the built-in and an identifying prefix for that namespace (file hints are not necessary to identify an imported built-in library).

7 Open Travel Projects

An Open Travel project (.otp) file describes a logical grouping of libraries that may or may not be related by include/import directives. A project grouping can contain a subset of the overall Open Travel model that is to be treated as a single compilation unit.

Each library that is contained within an Open Travel project is known as a project item. Library files that reside on the same local file system as the project itself are known as unmanaged project items. Unmanaged libraries MUST be referenced using an absolute file path or a relative path from the location of the referencing project file. Libraries that are stored in an OTM repository are managed project items that MUST be accessed via the OTM repository web service API's (see the Referenced Documents for more information).

The following XML sample contains an Open Travel project with a mix of managed and unmanaged libraries:

Example: Sample Open Travel Project (OTP)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project xmlns=" http://www.OpenTravel.org/ns/OTA2/Project_v01_00">
  <projectId>http://opentravel.org/ns1</projectId>
  <name>Sample Project</name>
  <description>Project description...</description>
  <ManagedProjectItem defaultItem="true">
    <Repository>OTA2_Repository</Repository>
    <BaseNamespace>http://opentravel.org/ns1</BaseNamespace>
    <Filename>ManagedLibrary_1_3_0.otm</Filename>
    <Version>1.3.0</Version>
  </ManagedProjectItem>
  <UnmanagedProjectItem>
    <FileLocation>libraries/UnmanagedLibrary.otm</FileLocation>
  </UnmanagedProjectItem>
  <RepositoryReferences>
    <RepositoryRef
        repositoryId="Sabre_STL2_Repository">
        http://opentravel.org:81/ota2-repository-service
    </RepositoryRef>
  </RepositoryReferences>
</Project>
```

In the above example, it should be noted that connectivity information is provided for the OTM repositories that are referenced by the managed libraries. Processors SHOULD specify this information so that new processor users can resolve connections to the repositories, even if those users have never accessed those repositories on prior occasions.

8 Open Travel Library Constructs

The constructs described in this section are the components or building blocks of the library terms described in later Chapter 9 of this specification.

8.1 Context Declarations

Context declarations provide a mechanism for identifying a system, domain, namespace, or situation with which other parts of an OTM model may be associated. For example, the XML example values for an OTM attribute may differ depending on whether the context of the example is for Air Transportation or Rail. Any entity that can reference a context declaration is said to be *context-sensitive*.

Context declarations are considered local to the OTM library in which they are defined, and MUST NOT be referenced by context-associated entities in other libraries. All context declarations support the following properties:

Property Name	Property Type	Description
Context	String	A short identifier (similar to a namespace prefix) that is used to identify the context within the library in which it is defined. Context identifiers MUST be unique to the library that declares the context.
Application Context	String	An identifier that indicates the system, domain, namespace, or situation to which the context applies. Application context identifiers SHOULD typically be expressed as fully qualified URI's.
Documentation	Documentation	Optional documentation for the context declaration.

The following snippet provides an example of context declarations that might be found in an OTM library:

Example: Context Declaration Samples

```
<Context context="air"
    applicationContext="http:/opentravel.org/ota2/air"/>
<Context context="rail"
    applicationContext="http:/opentravel.org/ota2/rail"/>
```

8.2 Documentation

The documentation element enables schema designers to provide annotations and references for various aspects of an OTM model. All documentation elements MAY contain the following properties:

Property Name	Property Type	Description
Description	String_Documentation	The definitive description of the OTM entity
Implementer	String_Documentation	Implementer-specific note(s) and other textual information for use by schema designers and application developers
Deprecated	String_Documentation	Notification(s) that the entity or term has been marked for deprecation and possible deletion in future versions
Reference	Any URI	URL(s) to reference information for the entity being documented
More Info	Any URI	URL(s) to additional external documentation that is not considered to be an official reference
Other Doc	Additional Doc	Other context-sensitive documentation item(s) not included in any of the other elements of the owning documentation

The following snippet provides an example of documentation content from an OTM library:

Example: OTM Documentation Sample

```
<Documentation>
    <Description>Description of the OTM model term.</Description>
    <Implementer>Implementation notes...</Implementer>
    <Deprecated>Scheduled for removal in version 2.5.</Deprecated>
    <Reference>http://www.OpenTravel.org/awg/ref.html</Reference>
    <MoreInfo>http://www.OpenTravel.org/awg/demo.html</MoreInfo>
    <OtherDoc context="air">Air transport documentation.</OtherDoc>
    <OtherDoc context="rail">Rail transport documentation.</OtherDoc>
</Documentation>
```

8.3 Equivalents

Equivalents provide a mechanism for designers to associate parts of an OTM model with an external application, schema, standard, or database. Equivalent mappings are context-sensitive, allowing multiple associations for a single OTM model element. When multiple equivalent mappings are provided, each one MUST be associated with a different context declaration.

Each equivalent is composed of the following properties:

Property Name	Property Type	Description
Context	String	The short identifier that associates this equivalent mapping with a particular context declaration.
Value	String	The value that indicates the details of the association with a particular context.

The following example depicts an attribute with multiple equivalent relationships:

Example: Attribute with Multiple Equivalent Mappings

```
<Attribute name="flightNumber" type="xsd:int">
  <Equivalent context="database">SCHEDULES_TABLE:FLT_NBR</Equivalent>
  <Equivalent context="xmlmsg">SchedRS/Flights/Flight#fn</Equivalent>
</Attribute>
```

8.4 Examples

Examples allow the schema designer to supply context-sensitive values for simple data types that are defined and/or referenced in an OTM model. When rendering sample output, processors SHOULD use these example values to define meaningful output.

Each example is composed of the following properties:

Property Name	Property Type	Description
Context	String	The short identifier that associates this example value with a particular context declaration.
Value	String	The example value. Processors SHOULD ensure that example values are value according to the rules of the simple type for which they are assigned.

The following OTM model snippet shows an attribute with multiple context-sensitive examples:

Example: Attribute with Multiple Examples

```
<Attribute name="stationCode" type="xsd:string">
  <Example context="air">DFW</Equivalent>
  <Example context="rail">DAL</Equivalent>
</Attribute>
```

8.5 Attributes

Attributes define named values for simple data types in an OTM model that support the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the attribute.
Type	Name_Type	Local or qualified name of the OTM term that defines the type of the attribute. If a local name is specified, processors MAY assume that the namespace of the type reference is the same as that of the attribute's owning term.
Mandatory	Boolean	Indicates whether the existence of the attribute is mandatory.
Documentation	Documentation	Optional attribute documentation provided by the model designer.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the attribute.
Example	Example	List of zero or more context-sensitive example values for the attribute.

Under normal conditions, attributes MUST only reference simple data types (e.g. simples and closed enumerations). In the following special circumstances, however, attributes type references MUST be considered valid even when referencing complex OTM types:

1. If the owner of an attribute is a value with attributes (VWA), other valid attribute types MUST include open enumerations or VWA's.

2. If the attribute's type reference is a top-level core object, the actual type of the attribute MUST be considered to be the core's Simple Facet type. In these cases, the core object MUST declare a simple facet type that is not '`ota:Empty`' (see section 9.5 for details).
3. List facets MUST only be referenced as an attribute type if the data type of the list facet is a core object's simple facet.

The following OTM library excerpt provides a number of valid attribute declarations:

Example: Attribute Declarations

```
<Attribute name="foo" type="LocalSimpleType" mandatory="false"/>

<Attribute name="bar" type="bar:BarSimpleType" mandatory="true"/>

<Attribute name="stationCode" type="xsd:string">
  <Documentation>
    <Description>Station code attribute description...</Description>
  </Documentation>
  <Example context="air">DFW</Equivalent>
  <Equivalent context="database">LOCATION_TABLE:STATION</Equivalent>
</Attribute>
```

8.6 Elements

Elements in an OTM model define named values for simple or complex data types. OTM element definitions are composed of the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the element.
Type	Name_Type	Local or qualified name of the OTM term that defines the type of the element. If a local name is specified, processors SHOULD assume that the namespace of the type reference is the same as that of the element's owning term.
Reference	Boolean	Indicates whether the term is to be contained by reference or by value (default is false). References MUST NOT be allowed for elements that are assigned to simple data types.
Mandatory	Boolean	Indicates whether the existence of at least one element is mandatory.
Repeat	Positive Integer or '*'	Indicates the maximum number of times that the element can be repeated. A '*' value MUST be specified to indicate unlimited repeats.
Documentation	Documentation	Optional element documentation provided by the model designer.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the element.
Example	Example	List of zero or more context-sensitive example values for the attribute. Examples SHOULD only be supplied for an element if its type reference is that of a simple data type.

In addition to the requirements set forth in the above list of properties, the following normative rules also apply to element type assignments:

1. If a property's type assignment references an empty standard or contextual facet (see sections 8.9 and 8.10 for details), processors MUST interpret the element's actual type assignment as that of its next-higher non-empty facet. For example, if an element references the empty detail facet of a business object, the actual type of the element will be interpreted to be the business object's summary facet (assuming the summary facet is not also empty). When this condition exists in an OTM model, processors SHOULD issue a WARNING notification for the user.
2. If a top-level business object (not one of its facets) is directly referenced by an OTM element, the reference MUST be interpreted to be valid for any of the business object's facets (excluding list and query facets).
3. If a top-level core object (not one of its facets) is directly referenced by an OTM element, the reference MUST be interpreted to be valid for any of the core object's facets (excluding list and simple facets).
4. If an element's type is considered to have an associated global XML element name, the 'name' property of the element MUST be ignored. The actual element name MUST match that of the global XML element name. If the 'name' property and the global element name do not match, processors SHOULD either automatically repair the nonconformance or issue a WARNING notification for the user.
5. If an element's owner extends an OTM term that declares another element with the same actual name, the extending term's owner MUST be interpreted as overriding the element from the extended term.
6. Considering the following scenario:
   - Term A2 extends term A1.
   - Term A1 declares an element E1 whose assigned type is T1.
   - Term A2 declares an element E2 whose assigned type is T2.

   Element E2 MUST be interpreted as overriding or eclipsing E1 if term T2 extends T1.

The following OTM library excerpt provides a number of valid element declarations:

Example: Element Declarations

```
<Element name="LocalCore_Summary" type="LocalCore_Summary"
    mandatory="false"/>

<Element name="bar" type="bar:BarSimpleType" mandatory="true">
  <Example context="air">BAR_VALUE</Equivalent>
</Element>

<Element name="Station" type="com:Station" mandatory="false"
    isReference="true" repeat="99">
  <Documentation>
    <Description>Station element description...</Description>
  </Documentation>
  <Equivalent context="database">LOCATION_TABLE:STATION</Equivalent>
</Element>
```

8.7 Indicators

Indicators in an OTM model represent a single named Boolean value that can be configured using the

following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the Boolean indicator.
Publish As Element	Boolean	Indicates whether the indicator SHOULD be represented as an attribute or element in a compiled schema for the OTM library (default is false).
Documentation	Documentation	Optional indicator documentation provided by the model designer.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the indicator.

The following library snippet provides some variations of valid indicator declarations:

Example: Indicator Declarations

```
<Indicator name="fooInd" publishAsElement="false">

<Indicator name="fooElementInd" publishAsElement="true">

<Indicator name="barInd">
  <Documentation>
    <Description>Indicator documentation...</Description>
  </Documentation>
  <Equivalent context="database">INDICATOR_TABLE:BAR</Equivalent>
</Indicator>
```

8.8 Named Entities

Names entities are a general classification for any OTM term or component that is considered *name-addressable*. Depending on the specific purpose and usage, name-addressable components can be references as attribute or element types or type extensions (see section 10). A named entity can be referenced from anywhere in an OTM model using its qualified name which is composed of a namespace URI and a local name identifier. In all cases, the namespace of a named entity is identifies by the namespace assignment of the library that owns the entity. The local name identifier is either derived or explicitly assigned, depending on the specific type of entity to which the name applies.

When addressing a named entity, one of two possible formats MUST be utilized:

- Local Addressing: Under this scheme, only the local name identifier is specified. The namespace is always assumed to be the same as that of the referencing term or entity.
- Qualified Addressing: Under this scheme, a qualified name is specified using a prefix and local name identifier in the format "`prefix:local-name`". The prefix MUST be a valid prefix identifier for an imported namespace (see section 6.1.2) within the library of the referencing term.

8.9 Standard Facets

Standard facets are a key underlying structure of most OTM models that act as containers for attributes, elements, and indicators. The following properties are available for all standard facets:

Property Name	Property Type	Description

Property Name	Property Type	Description
Type	Facet Type	Indicates the type of the facet; available types for standard facets are: • ID • Summary • Detail • Request • Response • Notification
Attributes	Attribute	A list of zero or more attribute declarations that are owned by the facet.
Elements	Element	A list of zero or more element declarations that are owned by the facet.
Indicators	Indicator	A list of zero or more indicator declarations that are owned by the facet.
Documentation	Documentation	Optional facet-specific documentation provided by the model designer.

Standard facets are named entities that MAY be used as type references for element declarations in an OTM model. The local name identity of a standard facet is a function of its owner's local name and the facet type as "`<owner-name>_<facet-type>`". For example the local name identity for the summary facet of the core object "`Foo`" would be "`Foo_Summary`".

The following OTM library excerpts provide a few examples of standard facet declarations. Notice that the facet type is implied by the tag name of the facet declaration itself.

Example: Standard Facet Declarations

```
<Summary>
  <Attribute name="foo" type="LocalSimpleType" mandatory="false"/>
  <Attribute name="bar" type="bar:BarSimpleType" mandatory="true"/>
  <Element name="LocalCore_Summary" type="LocalCore_Summary"
      mandatory="false"/>
</Summary>

<Detail>
  <Documentation>
    <Description>Detail facet documentation...</Description>
  </Documentation>
  <Element name="bar" type="bar:BarSimpleType" mandatory="true">
    <Example context="air">BAR_VALUE</Equivalent>
  </Element>
  <Indicator name="fooInd" publishAsElement="false">
  <Indicator name="fooElementInd" publishAsElement="true">
</Detail>
```

```
<Request>
  <Documentation>
    <Description>Request documentation...</Description>
  </Documentation>
  <Attribute name="stationCode" type="xsd:string">
    <Documentation>
      <Description>Station code attribute description...</Description>
    </Documentation>
    <Example context="air">DFW</Equivalent>
    <Equivalent context="database">LOCATION_TABLE:STATION</Equivalent>
  </Attribute>
  <Element name="Station" type="com:Station" mandatory=”false”
      isReference=”true” repeat=”99”>
    <Documentation>
      <Description>Station element description...</Description>
    </Documentation>
    <Equivalent context="database">LOCATION_TABLE:STATION</Equivalent>
  </Element>
  <Indicator name="barInd">
    <Documentation>
      <Description>Indicator documentation...</Description>
    </Documentation>
    <Equivalent context="database">INDICATOR_TABLE:BAR</Equivalent>
  </Indicator>
</Request>
```

8.10 Contextual Facets

Contextual facets are extensions of standard facets. In addition to the properties and characteristics of the standard facet specified in the previous section, contextual facets support labels and contextual references.

Property Name	Property Type	Description
Type	Facet Type	Indicates the type of the facet. For contextual facets, the only allowed types are: • Query • Custom
Context	String	Optional short identifier that associates this example value with a particular context declaration.
Label	String	Optional label for the facet.

The local name identity of a contextual facet is a function of its owner’s name, its facet type, and its context and label values. These rules vary slightly depending on the type of the contextual facet. The following table provides a complete set of examples that demonstrate the type-specific naming conventions that MUST apply for all contextual facets:

Facet Owner Name	Facet Type	Context	Label	Contextual Facet Name
FooTerm	Custom	<NULL>	“Web”	FooTerm_Web
FooTerm	Custom	“Air”	<NULL>	FooTerm_Air
FooTerm	Custom	“Air”	“Web”	FooTerm_Web
FooTerm	Custom	<NULL>	<NULL>	N/A (ERROR)
BarTerm	Query	<NULL>	“Web”	FooTerm_Query_Web
BarTerm	Query	“Air”	<NULL>	FooTerm_Query_Air
BarTerm	Query	“Air”	“Web”	FooTerm_Query_Web
BarTerm	Query	<NULL>	<NULL>	FooTerm_Query

The following OTM library excerpts provide a few examples of contextual facet declarations.

Example: Contextual Facet Declaration Examples

```
<Query context="profile" label="FindByUserId">
  <Attribute name="userId" type="xsd:string" mandatory="true"/>
</Summary>

<Custom context="" label="Web">
  <Documentation>
    <Description>Custom facet documentation...</Description>
  </Documentation>
  <Attribute name="foo" type="LocalSimpleType" mandatory="false"/>
  <Attribute name="bar" type="bar:BarSimpleType" mandatory="true"/>
  <Element name="Station" type="com:Station" mandatory="false"
      isReference="true" repeat="99">
    <Documentation>
      <Description>Station element description...</Description>
    </Documentation>
    <Equivalent context="database">LOCATION_TABLE:STATION</Equivalent>
  </Element>
  <Indicator name="fooInd" publishAsElement="false">
  <Indicator name="fooElementInd" publishAsElement="true">
</Custom>
```

8.11 Simple Facets

Simple facets specify the representation of a term or entity as a simple data type value. The following properties are available for simple facet declarations:

Property Name	Property Type	Description
Type	Name_Type	The simple data type assignment of the facet.
Documentation	Documentation	Optional facet documentation provided by the model designer.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the facet.
Example	Example	List of zero or more context-sensitive example values for the facet.

Simple facets are named entities whose local name identity is a function of its owner's local name as "`<owner-name>_Simple`". For example the local name identity for the simple facet of the core object "`Foo`" would be "`Foo_Simple`".

The following snippets demonstrate several variations of simple facet declarations.

Example: Simple Facet Declarations

```
<Simple type="LocalSimpleType"/>

<Simple type="bar:BarSimpleType"/>

<Simple type="com:StationCode">
  <Documentation>
    <Description>Simple facet description...</Description>
  </Documentation>
  <Example context="air">DFW</Equivalent>
  <Equivalent context="database">LOCATION_TABLE:STATION</Equivalent>
</Simple>
```

8.12 List Facets

List facets are always derived from other facet types, implying some repeating number of occurrences of the underlying facet. Unlike the other facet types, list facets are not explicitly declared in an OTM library. Instead, their existence is implied by the declaration of other facet types.

Like the other facet types, list facets are named entities. The local name identity of a list facet is derived from that of its underlying facet by simply appending "`List`" to the name. For example the local name identity for the summary list facet of the core object "`Foo`" would be "`Foo_Summary_List`".

8.13 Aliases

Aliases are named entities that serve as a key component of the "controlled vocabulary" aspect of OTA2.0 by defining alternative global names for a term or named entity. The sole property of an alias definition is its name.

Any entity that supports the definition of aliases MAY be referenced using the alias name instead of its primary assigned name. Aliases are always assigned to the same namespace as the term or entity that declares them. The local name identifier for an alias is its name.

The following OTM library snippet shows an example of an alias declaration for a core object:

Example: Alias Declarations

```
<CoreObject name="CoreWithAliases" notExtendable="true">
  <Aliases>Alias1 Alias2 Alias3</Aliases>
  ...
</CoreObject>

<CoreObject name="CoreWithoutAliases" notExtendable="true">
  <Aliases/>
  ...
</CoreObject>
```

In some cases, the existence of an alias is implied by its relationship with other terms or named entities. For example, the first core declaration in the above snippet would have three implied aliases for its summary facet called "`Alias1_Summary`", "`Alias2_Summary`", and "`Alias3_Summary`".

8.14 Roles

A role is used to define a purpose or usage characteristic of a term. For example, the term "`Phone`" can be further qualified on a per-usage basis using one of the roles "`Home`", "`Work`", or "`Cell`". Roles support the following properties:

Property Name	Property Type	Description
Value	String	The name of the role.
Documentation	Documentation	Optional role documentation provided by the model designer.

The following OTM library excerpts demonstrate some samples of valid role declarations:

Example: Role Declarations

```
<Role value="Home"/>

<Role value="Work">
  <Documentation>
    <Description>Work phone description...</Description>
  </Documentation>
</Role>
```

8.15 Enumeration Literals

Enumeration literals define the possible assignable values for the open and closed enumerations that declare them. Each enumeration literal supports the following properties:

Property Name	Property Type	Description
Literal	Enum_Literal_Value	The simple string that is assignable as a possible value for the enumeration that declares the literal.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the literal.
Documentation	Documentation	Optional role documentation provided by the model designer.

The following snippets demonstrate several variations of enumeration literal declarations.

Example: Enumeration Literal Declarations

```
<Value literal="MON"/>
<Value literal="TUE">
  <Documentation>
    <Description>Day of the week for Tuesday</Description>
  </Documentation>
</Value>
<Value literal="WED">
  <Documentation>
    <Description>Day of the week for Wednesday</Description>
  </Documentation>
  <Equivalent context="database">WEDNESDAY</Equivalent>
</Value>
```

9 Open Travel Library Terms

This chapter provides definitions and examples of the primary terms of the OTM modeling language. All of these terms extend, compose, or build upon the components defined in section 0 of this specification. Unless otherwise specified all of the terms described in this chapter should be considered type-addressable named entities.

9.1 Simple Types

Simple types are OTM terms that represent a simple data value (or list of values) that can be represented as a string literal within an XML message. Simple type declarations support the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the simple type.
Type	Name_Type	Named entity reference for the base type definition of the simple type. This reference MUST indicate a term or other named entity that represents a simple data type.
List Type Indicator	Boolean	Indicates whether this simple type represents a list of values of the term indicated by the 'type' property.
Pattern	String	Regular expression pattern that constrains the set of allowable values for the simple type. This property only applies to terms that are based (directly or indirectly) on the XML schema string type.
Max Length	Non-Negative Integer	Constrains the maximum length of possible string values for the simple type. This property only applies to terms that are based (directly or indirectly) on the XML schema string type.
Min Length	Non-Negative Integer	Constrains the minimum length of possible string values for the simple type. This property only applies to terms that are based (directly or indirectly) on the XML schema string type.
Fraction Digits	Non-Negative Integer	Constrains the maximum number of digits that MAY appear to the right of the decimal point for base-10 numeric values. This property only applies to terms that are based (directly or indirectly) on the XML schema decimal type.
Total Digits	Non-Negative Integer	Constrains the maximum total number of digits that in base-10 numeric values. This property only applies to terms that are based (directly or indirectly) on the XML schema decimal type.
Max Inclusive	String	Specifies the maximum inclusive value for values of the simple type. The value of this property MUST be in the value space of the base type reference.
Min Inclusive	String	Specifies the minimum inclusive value for values of the simple type. The value of this property MUST be in the value space of the base type reference.
Max Exclusive	String	Specifies the maximum exclusive value for values of the simple type. The value of this property MUST be in the value space of the base type reference.
Min Exclusive	String	Specifies the minimum exclusive value for values of the simple type. The value of this property MUST be in the value space of the base type reference.
Documentation	Documentation	Optional documentation provided by the model designer.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the simple type.
Example	Example	List of zero or more context-sensitive example values for the simple type.

The following OTM library excerpts provide several value simple type declarations:

Example: Simple Type Declarations

```
<Simple name="FooSimple" type="xsd:string">

<Simple name="AlphaNumericString" type="FooSimple"
    pattern=”[a-zA-Z0-9]+”>
  <Documentation>
    <Description>Alpha-numeric string value.</Description>
  </Documentation>
</Simple>

<Simple name="AlphaNumericStringList" type="AlphaNumericString"
    listTypeInd=”true”/>

<Simple name="Percentage" type="xsd:decimal" fractionDigits=”2”
    maxInclusive=”Z”>
  <Example context="default">25.50</Equivalent>
</Simple>

<Simple name="CapitalLetter" type="xsd:string" minInclusive=”A”
    maxInclusive=”Z”>
  <Equivalent context="IATA">ClassOfService</Equivalent>
</Simple>

<Simple name="Number1to10" type="xsd:int" minExclusive=”0”
    maxExclusive=”11”/>
```

9.2 Closed Enumerations

Closed enumerations are OTM terms that represent a closed collection of literal values. This means that the only possible values that can be assigned to an attribute or element of a closed enumeration type are the literal values defined in the enumeration itself (e.g. days of the week). Closed enumerations are composed of the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the enumeration term.
Values	Enumeration Literals	The list of literal declarations that define the allowable values for the enumeration.
Documentation	Documentation	Optional documentation provided by the model designer.

The following is an example of a valid closed enumeration declaration.

Example: Closed Enumeration Declaration

```
<Enumeration_Closed name=”DayOfWeek”>
  <Documentation>
    <Description>Days of the week.</Description>
  </Documentation>
  <Value literal="Monday"/>
  <Value literal="Tuesday"/>
  <Value literal="Wednesday"/>
  <Value literal="Thursday"/>
  <Value literal="Friday"/>
  <Value literal="Saturday"/>
  <Value literal="Sunday"/>
</Enumeration_Closed>
```

9.3 Open Enumerations

Open enumerations are OTM terms that represent an open collection of literal values. This means that the literals defined for an open enumeration are the most commonly used values, but do not comprise the set of all possible literals (e.g. airport code or passenger type code).

Open enumeration properties are exactly the same as those of closed enumerations. The allowance of undefined literal values is implied by the declaration of an open enumeration term, but the additional value field itself is not directly represented in the OTM model.

The following is an example of a valid open enumeration declaration.

Example: Open Enumeration Declaration

```
<Enumeration_Open name="PassengerTypeCode">
  <Documentation>
    <Description>Passenger type code values.</Description>
  </Documentation>
  <Value literal="ADT"/>
  <Value literal="C11"/>
  <Value literal="INF"/>
  <Value literal="MIL"/>
  <Value literal="SRC"/>
  ...
</Enumeration_Open>
```

9.4 Values with Attributes

A Value with Attributes (VWA) is an OTM term that describes a simple data type with associated attribute and/or indicator values. This definition implies that the VWA's are complex data types, but they are generally not governed by the naming and controlled vocabulary requirements that apply to other complex type terms. VWA's are typically used to define simple types whose values are further qualified by other simple data values (see examples in this section).

VWA's support the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the VWA term.
Type	Name_Type	Named entity reference for the base type definition of the VWA. This reference MUST indicate a term or other named entity that represents a simple data type or another VWA. Processors MUST interpret an assigned base type of 'ota:Empty' as a VWA without an assigned base type.
Attributes	Attribute	A list of zero or more attribute declarations that are owned by the VWA.
Indicators	Indicator	A list of zero or more indicator declarations that are owned by the VWA.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the base simple type of the VWA.
Example	Example	List of zero or more context-sensitive example values for the base simple type of the VWA.

Property Name	Property Type	Description
Value Documentation	Documentation	Optional documentation for the base type assignment for the VWA provided by the model designer.
Documentation	Documentation	Optional documentation for the VWA provided by the model designer.

The following OTM library excerpts provide several valid examples of VWA declarations.

Example: VWA Declarations

```
<ValueWithAttrs name="Amount" type="xsd:decimal">
  <Documentation>
    <Description>Documentation for the Amount VWA.</Description>
  </Documentation>
  <ValueDocumentation>
    <Description>Base type documentation...</Description>
  </ValueDocumentation>
  <Equivalent context="test">testenv/base-amount</Equivalent>
  <Example context="test">19.95</Example>
  <Attribute name="currency" type="ota:Code_Currnecy"/>
  <Indicator name="indicator1" publishAsElement="false"/>
</ValueWithAttrs>

<ValueWithAttrs name="VerifiedAmount" type="Amount">
  <Indicator name="verified"/>
</ValueWithAttrs>

<ValueWithAttrs name="SpecialConditionIndicators" type="ota:Empty">
  <Attribute name="wheelchairType" type="foo:WheelchairType"/>
  <Attribute name=”mealPrice” type=”VerifiedAmount”/>
  <Indicator name="wheelchairInd"/>
  <Indicator name="smokingInd"/>
  <Indicator name="mealPreferenceInd"/>
  <Indicator name="serviceAnimalInd"/>
</ValueWithAttrs>
```

9.5 Core Objects

Core objects are OTM terms that define complex data types that MAY also have simple type representations. The naming of core objects is governed by the OTM rules for controlled vocabulary, meaning that element with core object type assignments can only be assigned the name of the core object itself or one of its aliases. Core objects are typically used to declare terms that contain structured content, but do not possess a unique identity within any system or context.

The following properties are supported by core object declarations:

Property Name	Property Type	Description
Name	Name_XML	The name of the core object term.
Extension	Name_Type	Optional named entity reference to the core object that is extended by the one being declared. If present, the referenced entity MUST be another core object declaration. See section 10 for more information on OTM extensions and inheritance.
Aliases	Alias	List of zero or more aliases that define the allowable names by which the core object MAY be referenced.

Property Name	Property Type	Description
Not Extendable	Boolean	Indicates whether or not the core object supports ad-hoc extension points such as Extension Point Facets (see section 9.9). This value does not indicate whether or not values are allowed for the 'extension' property of a core object.
Simple Facet	Simple Facet	Indicates the simple representation of the core object. If the core object does not have a simple type representation, the simple facet SHOULD be blank or reference the 'ota:Empty' type.
Summary Facet	Standard Facet	Facet that contains the summary-level attribute, indicator, and element declarations for the core object. The summary facet of a core object MUST contain at least one member declaration.
Detail Facet	Standard Facet	Facet that contains the detail-level attribute, indicator, and element declarations for the core object. The detail facet of a core object MAY be empty.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the core object.
Documentation	Documentation	Optional documentation for the core object provided by the model designer.

Processors MUST interpret the facet hierarchy of a core object such that the detail facet is considered an extension of the summary facet contents.

The following OTM snippets provide several valid examples of core object declarations.

Example: Core Object Declarations

```
<CoreObject name="PersonName" notExtendable="true">
  <Aliases/>
  <Simple type="ota:Empty"/>
  <Summary>
    <Attribute name="firstName" type="xsd:string"/>
    <Attribute name="middleName" type="xsd:string"/>
    <Attribute name="lastName" type="xsd:string"/>
  </Summary>
  <Detail/>
</CoreObject>

<CoreObject name="PhoneNumber" notExtendable="false">
  <Aliases>Phone Telephone</Aliases>
  <Simple type="xsd:string"/>
  <Summary>
    <Attribute name="cityCode" type="foo:NumericString"/>
    <Attribute name="prefix" type="foo:NumericString"/>
    <Attribute name="lineNumber" type="foo:NumericString"/>
    <Indicator name="doNotCallInd"/>
  </Summary>
  <Detail/>
</CoreObject>
```

```
<CoreObject name="InternationalPhoneNumber" notExtendable="false">
  <Extension extends="PhoneNumber"/>
  <Aliases/>
  <Simple type="xsd:string"/>
  <Summary/>
  <Detail>
    <Attribute name="countryCode" type="foo:NumericString"/>
  </Detail>
</CoreObject>

<CoreObject name="Address" notExtendable="false">
  <Aliases/>
  <Simple type="xsd:string"/>
  <Summary>
    <Attribute name="street1" type="xsd:string"/>
    <Attribute name="street2" type="xsd:string"/>
    <Attribute name="city" type="xsd:string"/>
    <Attribute name="stateOrProvince" type="Enum_StateOrProvince"/>
    <Attribute name="postalCode" type="xsd:string"/>
  </Summary>
  <Detail>
    <Attribute name="country" type="iso:Code_Country"/>
    <Attribute name="attentionTo" type="xsd:string"/>
  </Detail>
  <Roles>
    <Role value="Home"/>
    <Role value="Work"/>
    <Role value="Shipping"/>
    <Role value="Billing"/>
  </Roles>
</CoreObject>
```

9.6 Business Objects

Business objects are OTM terms that define complex data types that exist as clearly defined business concepts that can be uniquely identified within an enterprise or business domain. Like core objects, business objects are governed by the OTM rules for controlled vocabulary, meaning that element with core object type assignments can only be assigned the name of the business object itself or one of its aliases.

The following properties are supported by business object declarations:

Property Name	Property Type	Description
Name	Name_XML	The name of the business object term.
Extension	Name_Type	Optional named entity reference to the business object that is extended by the one being declared. If present, the referenced entity MUST be another business object declaration. See section 10 for more information on OTM extensions and inheritance.
Aliases	Alias	List of zero or more aliases that define the allowable names by which the business object MAY be referenced.
Not Extendable	Boolean	Indicates whether or not the business object supports ad-hoc extension points such as Extension Point Facets (see section 9.9). This value does not indicate whether or not values are allowed for the 'extension' property of a business object.

Property Name	Property Type	Description
ID Facet	Standard Facet	Facet that defines the unique identity of the business object using the combination of its attribute, indicator, and element declarations. The ID facet of a business object MUST contain at least one attribute or element declaration.
Summary Facet	Standard Facet	Facet that contains the summary-level attribute, indicator, and element declarations for the business object. The summary facet of a business object MAY be empty.
Detail Facet	Standard Facet	Facet that contains the detail-level attribute, indicator, and element declarations for the business object. The detail facet of a business object MAY be empty.
Custom Facet	Contextual Facet	Zero or more contextual facets that define some number of attribute, indicator, and element declarations in addition to the ones defined in the summary facet of the business object.
Query Facet	Contextual Facet	Zero or more contextual facets, each of which defines a search function for the business object.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the business object.
Documentation	Documentation	Optional documentation for the business object provided by the model designer.

Processors MUST interpret facet hierarchy of a business object in the following manner:

- The summary facet MUST be interpreted as an extension of the ID facet.
- The detail facet MUST be interpreted as an extension of the summary facet.
- Any custom facets MUST be interpreted as extensions of the summary facet.
- Any query facets MUST be interpreted as not extending any other facet.

The above statements that imply an inheritance relationship mean that that the full list of declarations for the extending facet MUST include all members declared and inherited by the extended facet. For ordering purposes, all items inherited or declared by the extended facet MUST be interpreted to occur prior to any of the declarations inherited or declared by the extending facet.

The OTM excerpts below provide several valid examples of business object declarations.

Example: Business Object Declarations

```
<BusinessObject name="Profile" notExtendable="false">
  <Aliases>Customer Traveler Passenger</Aliases>
  <ID>
    <Attribute name="id" type="xsd:ID" mandatory=”false”/>
    <Element name="profileId" type="com:GUID" mandatory=”true”/>
  </ID>
  <Summary>
    <Element name=”name” type=”com:PersonName”/>
    <Element name=”homeAddress” type=”com:Address”/>
    <Element name=”phoneNumbers” type=”com:Phone_Summary_List”/>
  </Sumary>
  <Detail>
    <Indicator name=”loyaltyMember”/>
    <Element name=”loyaltyAccount” type=”com:AccountNumber”
        repeat=”99”/>
  </Detail>
  <Query context=”default” label=”FindByProfileId”/>
    <Element name="profileId" type="com:GUID" mandatory=”true”/>
  </Query>
</BusinessObject>

<BusinessObject name="CompanyXYZProfile" notExtendable="true">
  <Extension extends="Profile"/>
  <Aliases>LoyaltyInfo</Aliases>
  <ID/>
  <Summary/>
  <Detail/>
  <Query context=”xyz” label=”FindByLastName”/>
    <Element name=”seatingPreference” type=”xyz:SeatPreference”/>
  </Query>
  <Custom context=”xyz” label=”Web”/>
    <Element name=”seatingPreference” type=”xyz:SeatPreference”/>
  </Custom>
  <Custom context=”xyz” label=”CallCenter”/>
    <Element name=”billingAddress” type=”com:Address”/>
    <Element name=”seatingPreference” type=”xyz:SeatPreference”/>
  </Custom>
</BusinessObject>
```

9.7 Operations

Unlike the other components described in this section, operations are not first-class terms that are defined within an OTM library. Instead, operations are declared by services (see section 9.8) in order to define the functional actions of an OTM model. In spite of the fact that they are not first-class terms, operations are named entities that are governed by the OTM rules for controlled vocabulary.

OTM operations support the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the operation.
Extension	Name_Type	Optional named entity reference to the operation that is extended by the one being declared. If present, the referenced entity MUST be another operation declaration. See section 10 for more information on OTM extensions and inheritance.
Not Extendable	Boolean	Indicates whether or not the operation supports ad-hoc extension points such as Extension Point Facets (see section 9.9). This value does not indicate whether or not values are allowed for the 'extension' property of an operation.
Request Facet	Standard Facet	Facet that defines the request for the operation as a combination of its attribute, indicator, and element declarations. If the request facet of an operation is empty, processors MUST be assumed the request to be undefined.
Response Facet	Standard Facet	Facet that defines the response for the operation as a combination of its attribute, indicator, and element declarations. If the response facet of an operation is empty, processors MUST be assumed the response to be undefined.
Notification Facet	Standard Facet	Facet that defines a notification element for the operation as a combination of its attribute, indicator, and element declarations. If the notification facet of an operation is empty, processors MUST be assumed the notification to be undefined.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the operation.
Documentation	Documentation	Optional documentation for the operation provided by the model designer.

The following OTM library snippets provide a number of valid examples of operation declarations.

Example: Operation Declarations

```
<Operation Name="GetProfile" notExtendable="false">
  <Request>
    <Element name="ProfileQueryFindByProfileId" mandatory="true"
        type="Profile_Query_FindByProfileId"/>
  </Request>
  <Response>
    <Element name="ProfileDetail" repeat="0" type="Profile_Detail"/>
  </Response>
  <Notification/>
</Operation>
```

```
<Operation Name="FindXYZProfile" notExtendable="false">
  <Request>
    <Element name="CompanyXYZProfileQueryFindByLastName"
        type="CompanyXYZProfile_Query_FindByLastName"
        mandatory="true"/>
  </Request>
  <Response>
    <Element name="CompanyXYZProfileDetail" repeat="0"
        type="CompanyXYZProfile_Detail"/>
  </Response>
  <Notification/>
</Operation>
```

9.8 Services

Service declarations act as containers for the OTM operations described in the previous section. Unlike the other OTM library terms, only one service can be declared within a user-defined library. Another difference is that services are generally not considered to be name-addressable entities that can be referenced by other terms.

Service declarations support the following properties:

Property Name	Property Type	Description
Name	Name_XML	The name of the service.
Operations	Operation	One or more operation declarations for the service.
Equivalent	Equivalent	List of zero or more context-sensitive equivalent mappings for the service.
Documentation	Documentation	Optional documentation for the service provided by the model designer.

When services are defined in multiple minor versions of an OTM library's major version chain (e.g. 1.0, 1.1, 1.2, etc.), the later version(s) of the service SHOULD be considered to extend the prior service(s) defined in earlier versions of the OTM library. For this reason, all service definitions within a library's major version chain MUST be assigned the same name.

For the operation declarations within a service, any operations with the same name as one declared in earlier versions of a service SHOULD be interpreted as overriding the earlier version of the operation. New operations whose names do not match those in prior versions of the service SHOULD be considered additive to the service.

The following OTM snippet provides a valid service declaration example:

Example: Service Declarations

```
<Service name="ProfileService">
  <Documentation>
    <Description>Profile serivce description...</Description>
  </Documentation>
  <Operation Name="GetProfile" notExtendable="false">
    <Request>
      <Element name="ProfileQueryFindByProfileId" mandatory="true"
          type="Profile_Query_FindByProfileId"/>
    </Request>
    <Response>
      <Element name="ProfileDetail" repeat="0" type="Profile_Detail"/>
    </Response>
    <Notification/>
  </Operation>
```



```
  ...
  <Operation Name="CreateProfile" notExtendable="false">
    <Request>
      <Element name="ProfileDetail" mandatory=”true”
          type="Profile_Detail"/>
    </Request>
    <Response>
      <Element name="ProfileID" mandatory=”true” type="Profile_ID"/>
    </Response>
    <Notification/>
  </Operation>
</Service>
```

9.9 Extension Point Facets

Extension point facets are OTM terms that are not intended to stand alone as independent named entities. Instead, extension point facets supply additive content for existing terms without directly modifying those terms. This capability can be useful for implementing small changes to message structures without requiring a change for library terms that have been promoted to ‘Final’ status and are locked for editing.

Extension point facets support the following properties:

Property Name	Property Type	Description
Extension	Name_Type	Optional named entity reference to the operation that is extended by the one being declared. If present, the referenced entity MUST be another operation declaration. See section 10 for more information on OTM extensions and inheritance.
Attributes	Attribute	A list of zero or more attribute declarations that are owned by the extension point facet.
Elements	Element	A list of zero or more element declarations that are owned by the extension point facet.
Indicators	Indicator	A list of zero or more indicator declarations that are owned by the facet.
Documentation	Documentation	Optional documentation for the extension point facet provided by the model designer.

Extension point facets identify the standard or contextual facets they modify through their ‘extension’ property. For this reason, extension point facets are not considered named entities that can be referenced as independent OTM terms.

The following OTM library excerpts provide some valid examples of extension point facet declaration:

Example: Extension Point Facet Declarations

```
<ExtensionPointFacet>
  <Documentation>
    <Description>Extension point description...</Description>
  </Documentation>
  <Attribute name="xyzOfferId" type="foo:GUID"/>
  <Attribute name="productCategory" type="xsd:Enum_ProductCategory"/>
  <Indicator name="customOfferInd"/>
  <Extension extends="xyz:PricedOffer_Summary"/>
</ExtensionPointFacet>
```

```
<ExtensionPointFacet>
  <Element name="OfferInfoDetail" repeat="0"
      type="xyz:OfferInfo_Detail"/>
  <Extension extends="xyz:PricedOffer_Detail"/>
</ExtensionPointFacet>
```

9.10 XSD Schema Terms

XSD schema terms are OTM terms that represent type declarations from non-OTM XML schemas. There are three principle types of XSD schema types:

- XSD Simple Type (declared via "<xs:simpleType>")
- XSD Complex Type (declared via "<xs:complexType>")
- XSD Element (declared via a "<xs:element>" as a global XML element definition)

As named entities, all XSD schema types are eligible to be assigned as type references for OTM elements. Because it represents a simple data type, XSD simple type declarations can also be assigned as type references for attributes, OTM simple types, VWA value types, and simple facets for core objects.

The only OTM model property that MUST be present for an XSD schema term is its local name. There is no strict requirement for processors to support visibility into the definition and structure of legacy XSD terms as long as they can be referenced by other terms in the OTM model.

10 Extensions and Inheritance of Terms

OTM model constructs support two principle mechanisms that support inheritance: extensions and facet hierarchies. Several of the previous sections of this document have already discussed these concepts in a very general manner. The purpose of this chapter is to describe these topics in detail, including their specific effects on the inheritance of declarations between terms and facets.

Extensions:

The extension of terms exists when one term explicitly references another term of the same type via its Extension property. Processors MUST handle extensions of terms according to the following rules:

1. For any given term "B" that extends another term "A", both terms MUST be of the same general type (e.g. core object, business object, operation, etc.) in order for the extension to be valid.
2. When considering the inheritance between the facets of these terms, processors MUST interpret the inheritance of member attribute, indicator, and element declarations in the following manner:
    a. A facet of term "B" MUST be interpreted to inherit all of the member declarations from the corresponding term "A" facet of the *same type* (see 2(c) and 2(d) below).
    b. All inherited member declarations from the term "A" facet MUST be interpreted to occur in sequence ahead of those declared directly in the term "B" facet.
    c. Standard facets from terms "A" and "B" are considered to be the same type if their facet type properties match.

d. Contextual facets from terms "A" and "B" are considered to be the same type if their facet type, context, and label properties match.

Facet Hierarchies:

Unlike extension relationships that are explicitly defined by the model designer, facet hierarchies are implied by the nature of the term that owns the facets. For example, section 9.6 defines the facet hierarchy of a business object such that the summary facet extends the ID facet, the detail facet extends the summary facet, etc.

Processors MUST interpret the inheritance rules for facet hierarchies as follows:

1. Consider an OTM term with facets "F1" and "F2" such that "F2" is lower in the facet hierarchy than "F1".
2. Facet "F2" MUST be interpreted to inherit all of the member declarations (attributes, indicators, and elements) from facet "F1".
3. All of the declarations inherited from "F1" MUST be interpreted to occur in sequence ahead of those declared directly by the "F2" facet.

When determining the sequence of inherited terms, processors MUST interpret the declarations inherited from an extension relationship as taking precedence over the inheritance rules of the facet hierarchy. In the case of a core object's detail facet, for example, this means that all members inherited or declared by the summary facet MUST be interpreted to occur prior to any of the declarations inherited or declared by the detail facet.

11 Versioning of Libraries and Terms

The OTM modeling language is somewhat unique in that versioning strategies are incorporated into the structure of the language itself. This allows OTM to effectively represent, not only the terms declared in a model, but the change history of those terms. To this end, three principle types of version changes are supported: major versions, minor versions, and patches.

11.1 Version Schemes

While all OTM models MUST follow the same basic approach and structure when managing versioned components, some level of flexibility is allowed in the schemes that are used for version numbering. Generally, speaking, all version schemes MUST support the unique identification of major, minor, and patch-level version numbers.

At a minimum, version schemes that are implemented by processors MUST cover the following functions:

1. Encode a version identifier into the path of a base namespace URI
2. Decode a version identifier from a library's target namespace URI
3. Decode the base namespace URI from a library's target namespace URI
4. Construct a version-specific namespace prefix from a library's target namespace URI
5. Decode the major, minor, and patch components from a version identifier string (e.g. the major version component of "`1.2.3`" would be "`1`")

6. Identify the default version identifier to be used for newly-created libraries
7. Identify the major-version namespace associated with a library's target namespace URI (e.g. the major-version namespace for "`http://foo.com/v1_2_3`" could be "`http://foo.com/v1`")
8. Identify the possible namespaces associated with the major version chain of a library.
    a. The major version chain of any version is a sequence of version numbers (or version-encoded namespace URI's) that span from the current version to all prior versions up to and including the original major version of the chain.
    b. Example: The major version chain for "`http://foo.com/v3_2_2`" would include the following namespace URI's:

```
http://foo.com/v3_2_2
http://foo.com/v3_2_1
http://foo.com/v3_2_0
http://foo.com/v3_1_0
http://foo.com/v3_0_0
```

9. Implement the logic for incrementing and decrementing version identifiers at the major, minor, and patch version levels
10. Calculate the default filename hint for a library as a function of its target namespace, version identifier, and library name

At a minimum, processors `MUST` implement a default version scheme named "`OTA2`" that follows a standard numeric numbering pattern starting (by default) with version "`1.0.0`" for new libraries. In this default version scheme, the first digit indicates the major version number, the second digit indicates the minor version number, and the final digit indicates the patch level of a library.

11.2 Versioning of OTM Libraries

The versioning of user-defined OTM libraries is relatively straightforward. The version of a library is identified by its encoding into the target namespace URI (according to its version scheme – typically as the last component of the URI path). Since libraries act as containers for the terms and named entities of an OTM model, the version of each term is inherited from the library that contains it. Unlike user-defined libraries, legacy schemas are not required to maintain a version identifier as part of its target namespace URI.

When dealing with minor and patch versions, user-defined libraries have implied dependencies on the previous minor/patch version in the major version chain (see section 11.1, item #8) from which they were created.

The terms that can be defined in an OTM library are governed by the following rules, depending on its version type:

Major Versions

1. Any new term can be defined
2. If a prior version of the term existed, its content and structure can be modified in any way

Minor Versions

1. Any new term can be defined

2. Existing versioned terms (see section 11.3) can only be modified by adding indicators, optional attributes, or optional element declarations
3. Non-versioned terms cannot be modified in a minor version library

Patch Versions

1. Only new simple types, open/closed enumerations, and extension point facets can be defined
2. Extension point facets are only allowed to reference (extend) standard or contextual facets declared in a prior major or minor version of the patch library's major version chain

11.3 Versioning of OTM Terms

Although user-defined libraries can contain any type of term (within the constraints described in the previous section), only four of them are recognized as versioned OTM terms:

- Values with Attributes (VWA)
- Core Object
- Business Object
- Operation

Since none of these terms are allowed in patch version libraries, each of these terms only supports major and minor version levels. Patches for these terms and their facets are defined using extension point facets that are declared in a patch library version.

When a new minor version of a versioned term is declared, the previous version of the term is referenced via the new version's extension property. Since VWA's do not support extensions, the previous version of a VWA is referenced via its type property. For one term to be considered a later minor version of another term, all of the following conditions MUST be met:

1. The terms must be of the same type (business object, core, etc.) and have the same name
2. The terms MUST be declared in different libraries, and both libraries must have the same name, version scheme, and base namespace URI
3. The version of the extended term's library MUST be lower than that of the extending term's library version, but both libraries MUST belong to the same major version chain

Appendix A: Glossary

SIMPLE DATA TYPE – A type whose value may be expressed as a single lexical string value in an XML document with no associated attributes or child element tags

COMPLEX DATA TYPE – A type composed as a structure of simple values and/or nested complex type values

CONTEXT-SENSITIVE – Any term, named entity, or declaration that is associated with a context declaration in an OTM library

FACET HIERARCHY – An implied inheritance structure between the various types of facets that exist within an OTM term

VWA – Common abbreviation for an OTM Value with Attributes term

BASE NAMESPACE – A namespace URI that does not contain an encoded version identifier

Appendix B: Naming Conventions for XML Types and Elements

Although the OTM language specification is not specifically aligned with the XML schema language, the global naming conventions used in the language are closely aligned with the naming standards used when generating XML schemas. The following table provides a summary of the naming standards used for global type and element naming in XML schemas.

Term Name [Type]	Facet Type	Context	Label	Global XML Element Name	Global XML Type Name
MySimple [Simple]	N/A	N/A	N/A	N/A	MySimple
MyEnum [Closed Enum]	N/A	N/A	N/A	N/A	MyEnum
MyEnum [Open Enum]	N/A	N/A	N/A	N/A	MyEnum
MyVWA [VWA]	N/A	N/A	N/A	N/A	MyVWA
MyCore [Core Object]	N/A	N/A	N/A	MyCoreSubGrp	N/A
MyCore [Core Object]	Simple	N/A	N/A	N/A	MyCore_Simple
MyCore [Core Object]	Summary	N/A	N/A	MyCore [sub] MyCoreSummary [ns]	MyCore_Summary
MyCore [Core Object]	Detail	N/A	N/A	MyCoreDetail	MyCore_Detail
MyCore [Core Object]	Summary List	N/A	N/A	MyCore [sub] MyCoreSummary [ns]	MyCore_Summary
MyCore [Core Object]	Detail List	N/A	N/A	MyCoreDetail	MyCore_Detail
MyBO [Business Object]	N/A	N/A	N/A	MyBOSubGrp	N/A
MyBO [Business Object]	ID	N/A	N/A	MyBOID [sub] MyBOIdentity [ns]	MyBO_ID
MyBO [Business Object]	Summary	N/A	N/A	MyBO [sub] MyBOSummary [ns]	MyBO_Summary
MyBO [Business Object]	Detail	N/A	N/A	MyBODetail	MyBO_Detail
MyBO [Business Object]	Custom	N/A	Foo	MyBOFoo	MyBO_Foo
MyBO [Business Object]	Custom	Bar	N/A	MyBOBar	MyBO_Bar
MyBO [Business Object]	Custom	Bar	Foo	MyBOBarFoo	MyBO_Bar_Foo
MyBO [Business Object]	Query	N/A	N/A	MyBOQuery	MyBO_Query
MyBO [Business Object]	Query	N/A	Foo	MyBOQueryFoo	MyBO_Query_Foo

Term Name [Type]	Facet Type	Context	Label	Global XML Element Name	Global XML Type Name
MyBO [Business Object]	Query	Bar	N/A	MyBOQueryBar	MyBO_Query_Bar
MyBO [Business Object]	Query	Bar	Foo	MyBOQueryBarFoo	MyBO_Query_Bar_Foo
MyOp [Operation]	Request	N/A	N/A	MyOpRQ	MyOp_RQ
MyOp [Operation]	Response	N/A	N/A	MyOpRS	MyOp_RS
MyOp [Operation]	Notif.	N/A	N/A	MyOpNotif	MyOp_Notif

[sub] = Substitutable Element
[ns] = Non-Substitutable Element

Appendix C: Semantic Validation Rules for OTM Models

The tables provided in this appendix specify the normative semantic validation rules for each OTM component and term. In a number of cases, the same rule description applies to many different component types. To avoid duplication, these common rules have been consolidated in the "Common Validation Rules" section and are referenced by their Rule ID index in each of the applicable sections that follow.

Common Validation Rules

Rule ID (Index)	Validation Rule Description
REQUIRED VALUE	The property value is required. In the case of string values, empty strings are not allowed.
VALID NAME FORMAT	The format of the name, type reference, or other string value MUST conform to that of the simple type specified for the property in sections 6 – 9 of this document.
VALID NAME REFERENCE	The name MUST be formatted as "`localname`" or "`prefix:localname`"; the omission of the prefix indicates that the namespace is the same as that of the referencing term. In addition to being a properly formatted name, the referenced named entity MUST exist within the OTM model. In the case of extension reference, the extending entity MUST be the same type of term as the extended entity (e.g. core objects can only extend other cores).
NON-DEPRECATED TYPE REFERENCE	The referenced named entity type SHOULD not be deprecated. Deprecation is determined by the existence of a "Deprecation" element in the referenced entity's documentation element.
DUPLICATE DECLARATIONS NOT ALLOWED	The name or identity of a component or term declaration within its immediate owner/parent MUST NOT have a sibling component or term declared with the same name or identity.
DUPLICATE GLOBAL NAMES NOT ALLOWED	The qualified name of a term or named entity MUST be unique to the entire OTM model. Versioned terms are allowed to have duplicate names as long as they are considered valid minor versions of one another (see section 11.3).
CIRCULAR SIMPLE TYPE ASSIGNMENTS NOT ALLOWED	Type assignments that create direct or indirect circular references between simple type declarations are not allowed.
CIRCULAR EXTENSIONS NOT ALLOWED	Extension assignments that create direct or indirect circular references between terms are not allowed.
VALID MINOR VERSION EXTENSION	If a versioned entity extends another entity with the same name and same base namespace assignment, the version identifier of the extending entity MUST be later than the version identifier of the extended entity.
MAXIMUM ALLOWED LENGTH	The length of a string value MUST NOT exceed a specified maximum length.
VALID CONTEXT REFERENCE	The context ID reference MUST match the ID of a context declaration defined in the referring component's owning library.

1.1

Library Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Namespace	ERROR	• REQUIRED VALUE • MUST be a qualified (non-relative) namespace URI in URL format
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT
Version Scheme	ERROR	• REQUIRED VALUE • MUST be a valid version scheme identifier that is supported by the processor implementing this specification; all processors MUST support the "OTA2" version scheme
Prefix	ERROR	• REQUIRED VALUE
Include	ERROR	• Namespace of the included library MUST match the namespace of the library doing the include
Import	ERROR	• Namespace of the import declaration MUST NOT match the namespace of the library doing the import • Namespace of the imported library MUST match the namespace of the import declaration • The prefix associated with each imported namespace within a library must be unique the library in which the imports are declared
Terms	ERROR	• The qualified name (namespace + name) of each named entity in a library MUST be unique within the entirety of the OTM model • Only new simple types, open/closed enumerations, and extension point facets can be defined in patch library versions
Service	ERROR	• Duplicate service names MUST NOT be declared within different libraries assigned to the same namespace

OTM Project Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Namespace / Project ID	ERROR	• REQUIRED VALUE
Name	ERROR	• REQUIRED VALUE
Unmanaged Project Items	ERROR	• The location of an unmanaged project item (library or schema) MUST be a valid absolute file path or a relative path from the folder where the referencing project file is stored.

Property	Finding Type	Validation Rule Descriptions (or Index)
Managed Project Items	ERROR	• All fields that describe the ID the repository and the library resource within that repository MUST be specified • The repository that owns the managed resource MUST be accessible by the processor used to load the project

Context Declaration Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Context ID	ERROR	• REQUIRED VALUE • DUPLICATE DECLARATION NOT ALLOWED
Application Context	ERROR	• REQUIRED VALUE • DUPLICATE DECLARATION NOT ALLOWED

Documentation Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Description	ERROR	• MAXIMUM ALLOWED LENGTH = 10,000
Implementer	ERROR	• MAXIMUM ALLOWED LENGTH = 10,000
Deprecated	ERROR	• MAXIMUM ALLOWED LENGTH = 10,000
Reference	ERROR	• MUST be a valid URI string value
More Info	ERROR	• MAXIMUM ALLOWED LENGTH = 10,000
Other Doc	ERROR	• MAXIMUM ALLOWED LENGTH = 10,000 • VALID CONTEXT REFERENCE

Equivalent Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Context	ERROR	• REQUIRED VALUE • DUPLICATE DECLARATION NOT ALLOWED • VALID CONTEXT REFERENCE
Value	WARNING	• REQUIRED VALUE

Example Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Context	ERROR	• REQUIRED VALUE • DUPLICATE DECLARATION NOT ALLOWED • VALID CONTEXT REFERENCE
Value	WARNING	• MUST be valid according to the constraints of the simple type to which the example applies

Attribute Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE DECLARATIONS NOT ALLOWED
Type	ERROR	• REQUIRED VALUE • VALID NAME REFERENCE • Allowable type references are: Simple, Closed Enumeration, Simple Facet, XSD Simple Type. • VWA's and Open Enumerations are allowed type references if the owner of the attribute is a VWA. • Core objects are allowed type references if the core declares a non-empty simple facet. • List facet references are allowed if its underlying facet is a simple facet and the core object declares at least one role.
Type	WARNING	• NON-DEPRECATED TYPE REFERENCE • Warn for Boolean attributes that SHOULD be declared as indicators • Warn on usage of 'xsd:IDREF' or 'xsd:IDREFS'; SHOULD be declared as elements with a 'Reference' property value set to true.
Mandatory	ERROR	• Attributes defined for minor versions of a term MUST be optional.

Element Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE DECLARATIONS NOT ALLOWED
Name	WARNING	• Elements whose types have an associated global element name (see Appendix B) SHOULD be assigned that global element name (warn on name mismatch). • For elements whose Reference property is true and whose types are not associated with a global element name, the element name SHOULD end with "Ref".

Property	Finding Type	Validation Rule Descriptions (or Index)
Type	ERROR	• REQUIRED VALUE • VALID NAME REFERENCE • Allowable type references are: Simple, Closed Enumeration, Open Enumeration, VWA, Core Object, Business Object, Standard Facet, Contextual Facet, Simple Facet, List Facet, Alias, Role Enumeration, XSD Simple Type, XSD Complex Type, XSD Element • Element type assignments cannot create circular references in which all of the elements in the cycle are mandatory • Multiple elements that belong to the same inheritance hierarchy cannot be defined within the scope of its owning term (i.e. core or business object). The inheritance hierarchy includes a term and all of its facets, as well as any extended terms in the hierarchy. For the purposes of this rule, the inheritance hierarchy does not include aliases or their associated named entities. • For elements whose Reference property is true, the assigned type MUST be a complex type that declares an 'xsd:ID' attribute or element. • Elements MUST NOT reference the list facet of a core object unless the core object defines at least one role
Type	WARNING	• NON-DEPRECATED TYPE REFERENCE • Warn for Boolean elements that SHOULD be declared as indicators • Warn on usage of 'xsd:IDREF' or 'xsd:IDREFS'; SHOULD be declared as elements with a 'Reference' property value set to true • Warn if the assigned type is a standard or contextual facet with no assigned or inherited member declarations
Mandatory	ERROR	• Elements defined for minor versions of a term MUST be optional
Repeat	WARNING	• If the assigned type of an element is a list facet, the repeat value SHOULD be equal to the number of roles defined for the core object that owns the list facet.
Examples	WARNING	• Warn if examples are provided for an element whose assigned type is a complex data type

Indicator Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE DECLARATIONS NOT ALLOWED
Publish As Element	ERROR	• MUST be false if the owner of the indicator is a VWA

Standard Facet Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Facet Identity (owner + type)	ERROR	• DUPLICATE GLOBAL NAMES NOT ALLOWED
Facet Type	ERROR	• The facet type property MUST be valid for the owner of the facet declaration
Attributes & Elements	ERROR	• Only one attribute or element declared or inherited by a facet MAY be assigned the type 'xsd:ID'

Contextual Facet Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Facet Identity (owner + type + context + label)	ERROR	• DUPLICATE GLOBAL NAMES NOT ALLOWED
Facet Type	ERROR	• The facet type property MUST be valid for the owner of the facet declaration
Attributes & Elements	ERROR	• Only one attribute or element declared or inherited by a facet MAY be assigned the type 'xsd:ID'

Simple Facet Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Facet Identity (owner + type)	ERROR	• DUPLICATE GLOBAL NAMES NOT ALLOWED
Type	ERROR	• VALID NAME REFERENCE • CIRCULAR SIMPLE TYPE ASSIGNMENTS NOT ALLOWED • Allowable type references follow the same rules as attribute declarations • The simple facet for a later minor version of a core object cannot change its simple type assignment
Type	WARNING	• NON-DEPRECATED TYPE REFERENCE

Role Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE DECLARATIONS NOT ALLOWED

Enumeration Literal Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE DECLARATIONS NOT ALLOWED

Simple Type Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Type	ERROR	• REQUIRED VALUE • VALID NAME REFERENCE • Allowable type references are: Simple and XSD Simple Type • CIRCULAR SIMPLE TYPE ASSIGNMENTS NOT ALLOWED
Type	WARNING	• NON-DEPRECATED TYPE REFERENCE
List Type Indicator	ERROR	• A simple type MUST NOT be declared as a list if its type assignment is itself a simple list type
Pattern	ERROR	• MUST be a valid regular expression
Min Length	ERROR	• If present, MUST be greater than or equal to zero
Max Length	ERROR	• If present, MUST be greater than or equal to the Min Length value (or zero if a Min Length value is not defined)
Min Inclusive	ERROR	• If present, MUST be greater than or equal to zero
Max Inclusive	ERROR	• If present, MUST be greater than or equal to the Min Inclusive value (or zero if a Min Inclusive value is not defined)
Min Exclusive	ERROR	• If present, MUST be greater than or equal to zero
Max Exclusive	ERROR	• If present, MUST be greater than or equal to the Min Exclusive value (or zero if a Min Exclusive value is not defined)
Min/Max Length, Min/Max Inclusive, Min/Max Exclusive, Pattern, Fraction Digits, Total Digits	WARNING	• Warn if present when the List Type Indicator is true.

Closed Enumeration Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Values	ERROR	• At least one enumeration literal value MUST be defined

Open Enumeration Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Values	ERROR	• At least one enumeration literal value MUST be defined

Value with Attributes Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Type	ERROR	• VALID NAME REFERENCE • VALID MINOR VERSION EXTENSION • Allowable type references follow the same rules as attribute declarations (open enumerations and other VWA declarations are also allowed) • Circular references between VWA type assignments and/or VWA attribute declaration are not allowed • The simple facet for a later minor version of a core object cannot change its simple type assignment
Type	WARNING	• NON-DEPRECATED TYPE REFERENCE
Attributes & Indicators	WARNING	• At least one attribute or indicator MUST be declared for the VWA
Attributes	ERROR	• If a declared attribute has the same name as an attribute inherited from another VWA, the type assignments of both attributes MUST be identical • If the VWA simple type is an open enumeration, an attribute named '`extension`' cannot be declared or inherited • If one or more VWA attributes are typed as open enumerations, no attributes named '`<open-enum-attribute>Extension`' can be declared or inherited

Core Object Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Extension	ERROR	• VALID NAME REFERENCE • CIRCULAR EXTENSIONS NOT ALLOWED • VALID MINOR VERSION EXTENSION
Summary Facet	ERROR	• The summary facet of a core object MUST declare or inherit at least one attribute, element, or indicator

Business Object Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Extension	ERROR	• VALID NAME REFERENCE • CIRCULAR EXTENSIONS NOT ALLOWED • VALID MINOR VERSION EXTENSION
ID Facet	ERROR	• The ID facet of a business object MUST declare or inherit at least one attribute or element

Operation Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • DUPLICATE GLOBAL NAMES NOT ALLOWED
Extension	ERROR	• VALID NAME REFERENCE • CIRCULAR EXTENSIONS NOT ALLOWED • VALID MINOR VERSION EXTENSION
Request, Response & Notification Facets	ERROR	• The non-empty facets of an operation MUST conform to one of the following recognized enterprise messaging patterns: ○ One-Way (RQ only) ○ Notification (Notif only) ○ Request-Response (RQ + RS) ○ Solicit Notification (RQ + Notif) ○ Request-Response with Notification (RQ + RS + Notif)

Service Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Name	ERROR	• REQUIRED VALUE • VALID NAME FORMAT • If a service is declared in a prior minor version of the owning library, the name of the services MUST be identical
Operations	ERROR	• A service MUST declare or inherit at least one operation

Extension Point Facet Validation Rules

Property	Finding Type	Validation Rule Descriptions (or Index)
Extension	ERROR	• REQUIRED VALUE • VALID NAME REFERENCE • The extension point facet MUST be assigned to a different namespace than the extended term or named entity
Extension	WARNING	• Warn if the extended term or named entity is deprecated