

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.

Document Purpose:

This document is intended to provide an overview of the OTM Objects and some of the design goals the OpenTravel Model (OTM) is based on.

Introduction to OTM Objects User Guide

This guide provides an overview of the OpenTravel Model (OTM). OTM simplifies defining XML Schemas and messages that conform to the [OpenTravel 2.0 XML Schema Best Practices](#). For descriptions of the best practices for designing libraries of OTM objects, see the [Modeling OTM Objects Best Practices](#).

While not necessary to understand this guide it does assume you are familiar with reading XML and XML Schemas. Additionally, screen captures from the OTM-DE Designer tool are used to help illustrate the structure of the OTM model objects.

Contents

Simple Objects	3
<i>Simple Type</i>	3
— Constraints	3
— XML Representation	3
— Editor Representation	4
— Examples.....	4
<i>Closed Enumeration</i>	4

— XML Representation	4
— Editor Representation	4
Complex Objects	4
<i>Open Enumeration</i>	4
— XML Representation	5
— Editor Representation	5
<i>Value With Attributes</i>	5
— XML Representation	6
— Editor Representations	6
<i>Core Objects</i>	6
— Facets	6
— XML Representations	7
— Editor Representation	9
<i>Business Object</i>	10
— Facets	10
— XML Representations	10
— Editor Representations	11
Service	12
— XML Representation	12
— Editor Representation	12

Introduction

The OpenTravel Model (OTM) simplifies defining XML Schemas and messages that conform to the OpenTravel 2.0 XML Schema Best Practices. The model defines an OTM Library as a single file that contains metadata and definitions of model objects.

- Metadata
 - Version, namespace, name
 - Imports and includes
- Simple Objects
 - Simple Types
 - Closed Enumeration
- Complex Objects
 - Open Enumeration
 - Value With Attributes
 - Core Object
 - Business Object
- Service
 - Operations
 - Messages

Complex objects and messages are objects that define a collection of properties. Properties define where XML elements (tags) will be in an XML message. Properties represent XML elements or attributes.

Properties are named and assigned a type. The type can be a simple field or a complex object. Simple properties define XML elements or attributes that do not contain XML markup (tags).

The OpenTravel Model (OTM) is designed to be compiled by the OTM-DE Compiler into XML Schemas, XML examples and web service descriptions (WSDL). The OTM is an XML document defined by the OpenTravel Library Model schema. Throughout this document you will find illustrations where an OTM model object is compiled into its XML Schema representation.

Simple Objects

Simple objects represent XML Schema simple types. Simple types contain a single data value without any XML markup (attributes or child elements). There are two types of simple objects: the simple type and closed enumeration.

Simple Type

A simple type is named definition of an object with descriptions, other documentation, example values equivalents and constraints



— Constraints

The constraints begin with identifying a base type then add additional constraints. Often the base type will be a simple type defined in the XML Schema specification.

To the constraints associated with the base type, the user can add additional constraints. These include:

- **Pattern** – An expression that limits the valid character using the XML [regular expression syntax](#).
- **Min and max length** – the minimum and maximum number of characters allowed.
- **Fraction digits** – the maximum number of fraction digits allows
- **Total digits** – the total numeric characters allowed in a decimal based type.
- **Min/Max Inclusive/Exclusive** – sets the lower and upper bounds on the range of values allowed by the type.

— XML Representation

When compiled, a simple object becomes an XML Schema simpleType. All documentation and example values are copied into the type's annotation element.

Property Characteristics

The details that define a property.

Object Type

Simple Type

Namespace

http://opentravel.org/common/v02

Name

Code_IATA_City

Description

A three character code for a city.

Type

string

Type NS

xsd

Role

Element

Repeat

1

Mandatory

☐

Pattern

[a-zA-Z]{3}

Min Length

0

Max Length

0

Fraction Digits

-1

Total Digits

0

Min Inclusive

Max Inclusive

Min Exclusive

Max Exclusive

List

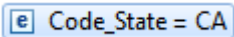
☐ List

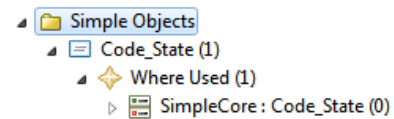
Example

Equivalent

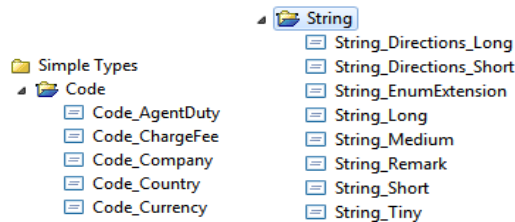
— Editor Representation

In the OTM-DE Editor navigator view the simple type is shown along with a *Where Used* child.

In the Example View the simple object is shown along with its example value. 



— Examples



Closed Enumeration

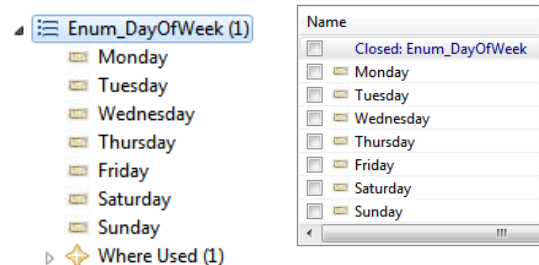
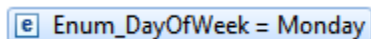
A closed enumeration is a list of values. The type and each value can have a description and other documentation. Closed enumerations values are guaranteed to not change without creating a new version of the model. They are best used to describe objects whose set of values does not change *or* whose set of values are important to manage because they are likely to significantly impact implementing applications.

— XML Representation

When compiled the closed enumeration becomes a simple type with enumerated values, complete with documentation.

— Editor Representation

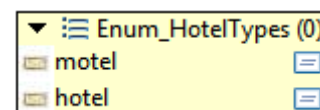
In the OTM-DE Editor, the enumeration is shown in the Navigator, Type and Example Views.



Complex Objects

Open Enumeration

Like a closed enumeration, open enumerations are a list of values. However Open Enumerations include



a value of “Other_” and an “extension” attribute. This provides a consistent approach to creating enumerations for lists of values that are preferred or common while still allowing alternative values.

— XML Representation

When compiled the open enumeration creates an XSD simpleType for the enumerated list and a complexType with simple content to add the extension attribute.

When used in an XML document an open enumeration with an extended value looks like:

```
<HotelType extension="cottage">Other_</HotelType>
```

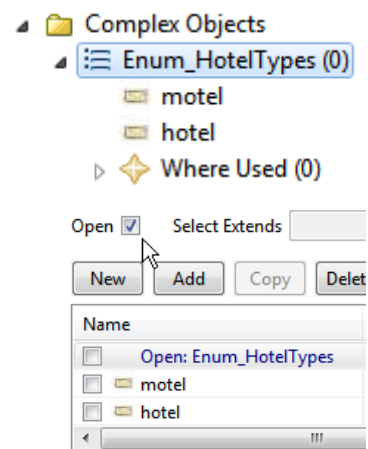
— Editor Representation

In the Editor, an open enumeration is listed under the Complex Objects folder because they have an attribute and can’t be used as a simple type.

Except for the “Open” check box being selected, the Type View displays open enumerations the same as closed enumerations.

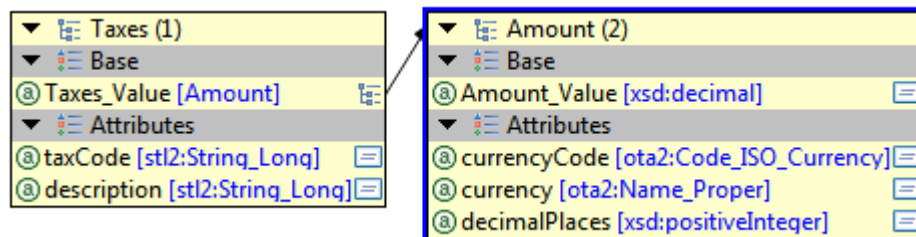
In Example View the value and extension are shown.

Enum_HotelTypes = motel
extension = Other_Value



Value With Attributes

The Value With Attributes (VWA) is a collection of attributes that relate to the value. For example the currency and currency code in the Amount VWA relate to the Amount_Value. VWAs can also have an Empty value in which case they are simply a group of attributes.



The value and attribute types of a VWA can be any simple type, open enumeration or other VWA.

For more on the VWA, see OTM-DE – UserGuide – ValueWithAttribute.docx.

— XML Representation

When compiled, a VWA becomes a single XSD complexType with simpleContent. The value is the base type of the simpleContent to which all attributes are added.

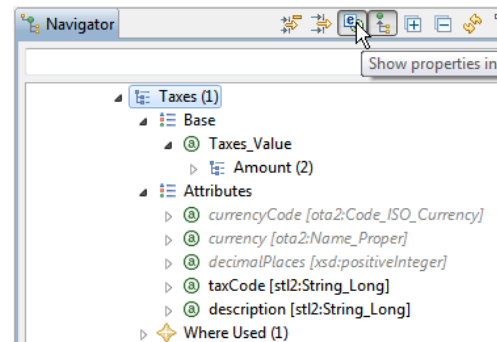
— Editor Representations

In the Navigator view you can see the base type which defines the value of the VWA and the attributes. With “Show Properties” selected, the types assigned to can also be seen and navigated into. With “Display Inherited Properties” selected attributes from Open Enumerations or VWA assigned as types will also be shown as attributes in a light-grey italic font.

In Type View, the VWA is displayed with two facets: the *Base* for the value type and the *Attributes* facets.

Name	Role	Type
Base		
ⓐ Taxes_Value	Simple	Amount
Attributes		
ⓐ taxCode	Attribute	stl2:String_Lor...
ⓐ description	Attribute	stl2:String_Lor...

Taxes = 35.68
 ⓐ currency = Euro
 ⓐ currencyCode = EUR
 ⓐ decimalPlaces = 2
 ⓐ description = AirportTax
 ⓐ taxCode = Tax123



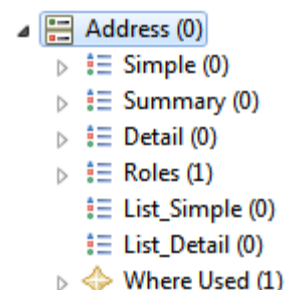
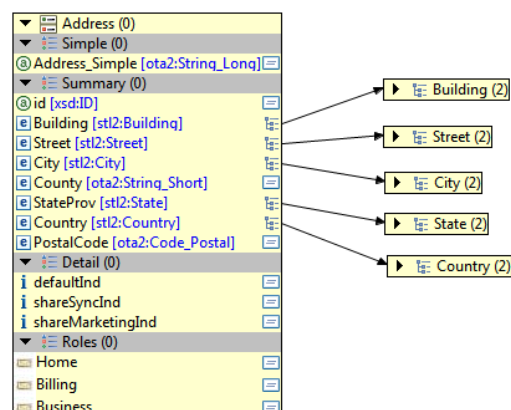
Core Objects

The Core Object is an object that describes multiple representations of the real-world object it describes. It is intended to describe real-world objects that are the same regardless of business context—an Address is an Address in searching for golf courses and booking an airline ticket.

— Facets

A Core Object defines up to six different representations that can be used as types in other objects.

1. **Simple** – a simple type. In the address example, this is a *long string* that can be used for an unstructured address.



2. **Summary** – a set of properties (indicators, attributes or elements)
3. **Detail** – a set of properties that extend the *summary* properties.
4. **Roles** – qualifiers that define the different roles the real-world object can have. For example an address can be home or business. When used as a type, roles become an open enumeration.
5. **Simple List** – an XML list of the simple representation.
6. **Detail List** – a repeating group of the detail facet. The detail facet includes a role attribute whose values are a closed enumeration of the defined roles. The group repeats once for each role defined.

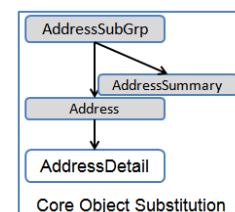
— XML Representations

A full Core Object with properties, types and roles all assigned compiles into:

- Four XSD simple types
 - ▷ `Address_Simple : ota2:String_Long`
 - ▷ `Address_Simple_List : ns5:Address_Simple`
 - ▷ `Enum_AddressRole_Base : ns5:Enum_AddressRole_Open`
 - ▷ `Enum_AddressRole_Open : xsd:string`
- Two XSD complex types
 - ▷ `Address`
 - ▷ `Address_Detail : ns5:Address`
- Four XML elements.
 - ▷ `Address : Address`
 - ▷ `AddressDetail : Address_Detail`
 - ▷ `AddressSubGrp : Address`
 - ▷ `AddressSummary : Address`

Core Objects create a *substitution group* that allows either the Summary or Detail facet to be used when the core object is assigned as a type. In the compiled XSD Schema the substitution group for Address looks like:

```
<xsd:element name="AddressSubGrp" type="Address"/>
<xsd:element name="Address" substitutionGroup="AddressSubGrp" type="Address"/>
<xsd:element name="AddressSummary" substitutionGroup="AddressSubGrp" type="Address"/>
<xsd:element name="AddressDetail" substitutionGroup="Address" type="Address_Detail"/>
```



The Core Object design allows the model designer the choice of assigning types using either “Address”, “AddressSummary” or “AddressDetail”.

- When “AddressSummary” or “AddressDetail” are used then the XML data must be of that type.
- When “Address” is used, the data can contain either an <Address>, <AddressSummary> or <AddressDetail> element.

When an OTM property type is set to “Address” the schema uses the substitution group head:
`<xsd:element ref="AddressSubGrp"/>`. The substitution group allows all three elements shown below to be valid in the XML data.

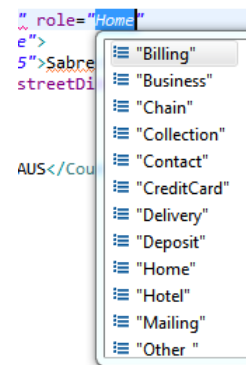
```
<Address id="address_1" role="Home">
  <Street>Market Drive</Street>
  <City>DAL</City>
  <StateProv>TX</StateProv>
  <Country>AUS</Country>
  <PostalCode>33626</PostalCode>
</Address>

<AddressSummary id="address_2" role="Business">
  <Street>Market Drive</Street>
  <City>DAL</City>
  <StateProv>TX</StateProv>
  <Country>AUS</Country>
  <PostalCode>33626</PostalCode>
</AddressSummary>

<AddressDetail defaultInd="true" id="address_3" role="Mailing" shareSyncInd="true">
  <Building bldgNumber="B" roomNumber="B3-445">Headquarters</Building>
  <Street pO_Box="4892" streetDirection="NW" streetNmbrSuffix="A">Market Drive</Street>
  <City name="DAL">Dallas</City>
  <StateProv name="Joe Smith">TX</StateProv>
  <Country alpha2Code="US" name="UnitedStates">US</Country>
  <PostalCode>33626</PostalCode>
</AddressDetail>
```

The enumerated roles assure that all users of the Core Object use the same set of values to qualify usage. When working with the XML data or classes created from the schema, the list of values for the “role” attribute is constrained and tools will often present that list with their “code completion” features.

```
<!--Roles Open Enumeration -->
<xsd:element name="Enum_AddressRole" type="Enum_AddressRole"/>
```



Here is an example of the compiled XML Schema and XML document created when an address simple list is used as a type:

```
<!--Simple List -->
<xsd:element name="Addresses" type="Address_Simple_List"/>
```

```
<Addresses>I am a long string. I am a long string. I am a long string.</Addresses>
```

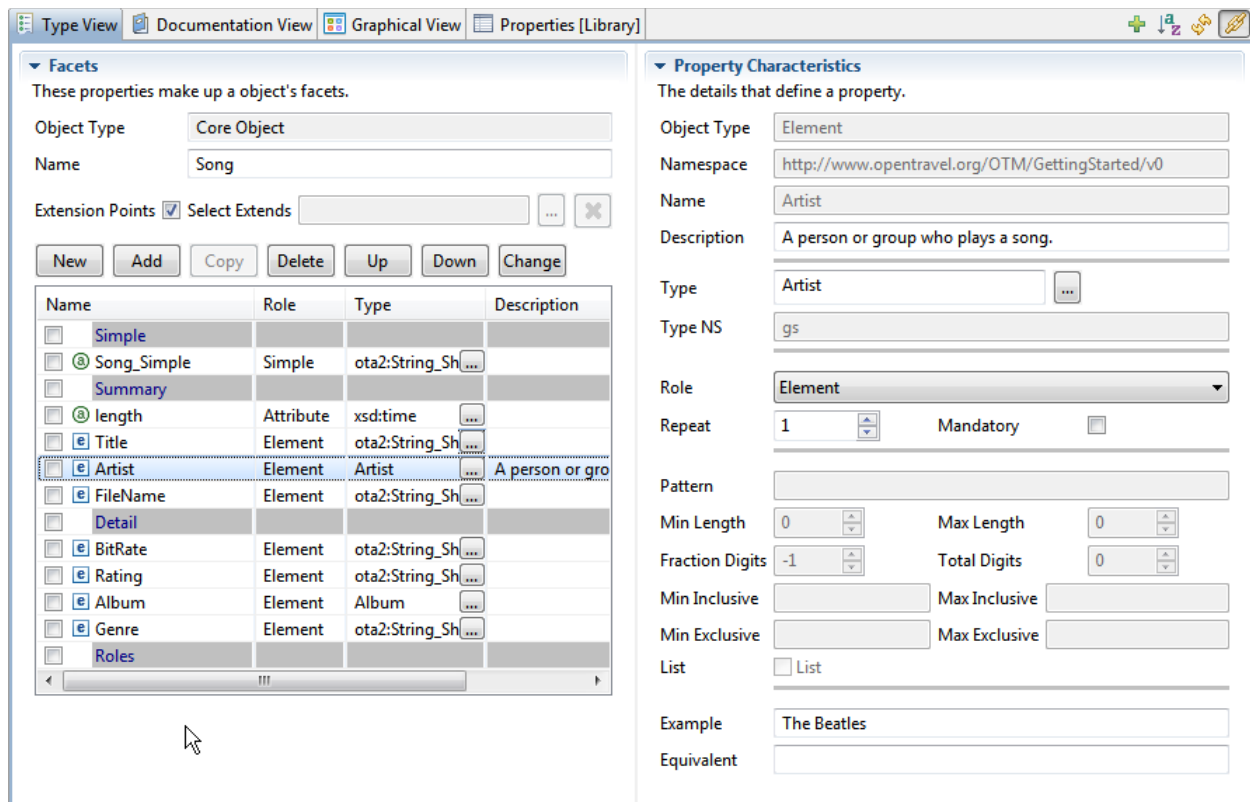
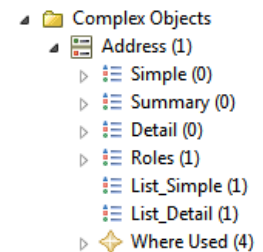

When the address detail list is used as a type it compiles into an XML Schema element that repeats once for each role represented by the maxOccurs value:

```
<!--Detail List -->
<xsd:element maxOccurs="15" minOccurs="1" ref="AddressDetail"/>
```

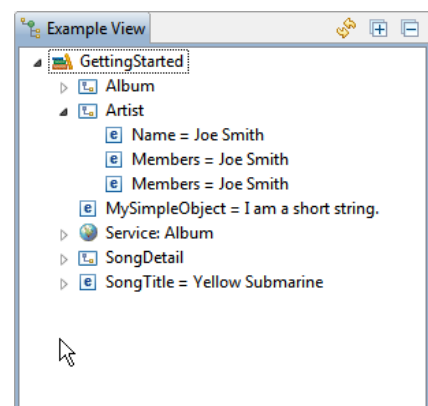
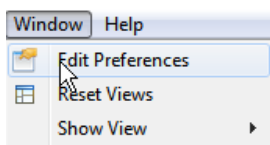
— Editor Representation

The Navigator View presents a Core Object with its facets as immediate children. These can be expanded to examine the properties of each facet.

The Type View is the primary editing view for complex objects. It presents the facets and properties of the Core Object. The user can add properties via Drag-n-Drop or using the Add wizard. The characteristics of each property can be edited in the right-hand panel for the selected property.

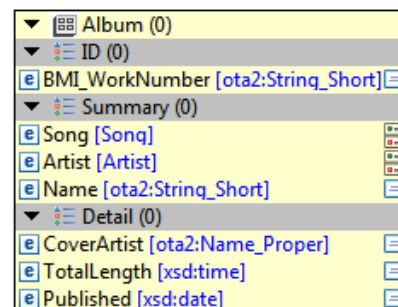


The Example View will show either the summary or detail facet depending on the setting in the compiler preferences.



Business Object

A Business Object defines multiple representations for a single real-world item or concept. All Business Objects have representations for identifying the item, summary and detail descriptions. In addition they can define the properties or sets of properties that can be used to query or find the items as well as custom sets of properties used in specific business contexts.



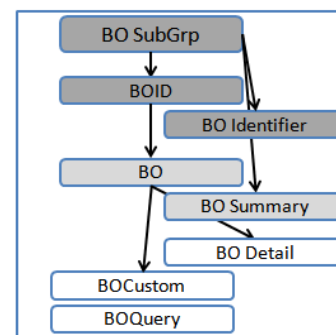
— Facets

1. **ID** - The issuer and bag tag number are used as unique identifiers for the bag.
2. **Summary** - The summary representation has an indicator and four attributes.
3. **Detail** - The detail adds two more properties to these four.
4. **Custom** – The “Lost” custom facet defines a representation to be used when the bag is lost.
5. **Query** – The query facet defines a representation that says that you can use bag tag number, carrier and description in queries.

— XML Representations

When compiled a Business Object will create a XSD complexType for each facet that has properties. In the example, five complex types are created.

- ▶ CheckedBag_Air : ns5:CheckedBag_Air_ID
- ▶ CheckedBag_Air_Detail : ns5:CheckedBag_Air
- ▶ CheckedBag_Air_ID
- ▶ CheckedBag_Air_Lost : ns5:CheckedBag_Air
- ▶ CheckedBag_Air_Query



The summary facet inherits the ID properties. The detail and custom facets inherit all of the summary properties.

When compiled the XSD schema created has an element for each facet plus one for the substitution group (CheckedBag_AirSubGrp), and one for the ID, summary and detail facet that are not substitutable. The non-substitutable elements are used when the model has a facet assigned as a property type.

```
<xsd:element name="CheckedBag_AirSubGrp" type="CheckedBag_Air_ID"/>
<xsd:element name="CheckedBag_AirID" substitutionGroup="CheckedBag_AirSubGrp"
  type="CheckedBag_Air_ID"/>
<xsd:element name="CheckedBag_AirIdentifier" substitutionGroup="CheckedBag_AirSubGrp"
  type="CheckedBag_Air_ID"/>
```

```
<xsd:element name="CheckedBag_Air" substitutionGroup="CheckedBag_AirID"
  type="CheckedBag_Air"/>
<xsd:element name="CheckedBag_AirSummary" substitutionGroup="CheckedBag_AirSubGrp"
  type="CheckedBag_Air"/>
<xsd:element name="CheckedBag_AirDetail" substitutionGroup="CheckedBag_Air"
  type="CheckedBag_Air_Detail"/>
<xsd:element name="CheckedBag_AirLost" substitutionGroup="CheckedBag_Air"
  type="CheckedBag_Air_Lost"/>

<xsd:element name="CheckedBag_AirQuery" type="CheckedBag_Air_Query"/>
```

— Editor Representations

The Editor presents a Business Object in the same way as a Core Object. The Navigator View shows the object's facets which can be expanded to display the properties. The Type View presents an editable table of facets and properties and the property characteristics. The Example View shows either summary or detail representations depending on the preference setting.

- Album (0)
 - ID (0)
 - Summary (0)
 - Detail (0)
 - Where Used (0)

The screenshot displays the OpenTravel Editor interface with two main panels:

Facets Panel: Shows the 'Album' Business Object. It includes a table of facets with columns for Name, Role, Type, and Description.

Name	Role	Type	Description
ID			
BMI_WorkNumber	Element	ota2:String_Sh...	
Summary			
Song	Element	Song	
Artist	Element	Artist	A pers...
Name	Element	ota2:String_Sh...	
Detail			
CoverArtist	Element	ota2:Name_Pr...	
TotalLength	Element	xsd:time	
Published	Element	xsd:date	

Property Characteristics Panel: Shows the details for the 'Published' property.

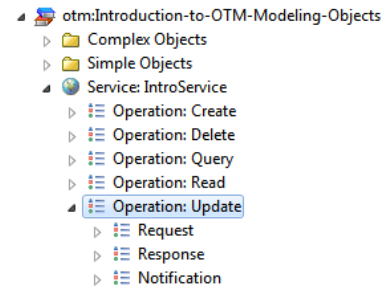
- Object Type: Element
- Namespace: http://www.opentravel.org/OTM/GettingStarte...
- Name: Published
- Description:
- Type: date
- Type NS: xsd
- Role: Element
- Repeat: 1, Mandatory
- Pattern:
- Min Length: 0, Max Length: 0
- Fraction Digits: -1, Total Digits: 0
- Min Inclusive: , Max Inclusive:
- Min Exclusive: , Max Exclusive:
- List: ☐ List
- Example:
- Equivalent:

Service

An OTM Library can define a service complete with 1 or more operations and Request, Response and Notification messages. Service messages are compiled into the XSD Schemas and the Operations and Service details are used to create the WSDL service description.

A library can only have one service. The user can delete notifications and/or responses if appropriate for their service interaction pattern.

Note, “REST” services can be defined for business objects without using the Service Object.



— XML Representation

Each request, response and notification message in the service compiles into an XML element and complex type.

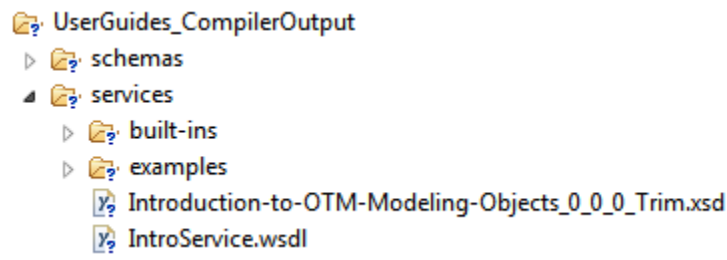
The element extends the OTA2 payload element of that message type. This allows applications and messaging infrastructure systems to unambiguously identify the actual message within an envelope such as a SOAP envelope.

```
<xsd:element name="CreateNotif"
  substitutionGroup="ota2msg:OTA2_Notif_Payload" type="Create_Notif"/>
```

To make the substitution legal, the complex type extends the generic message payload:

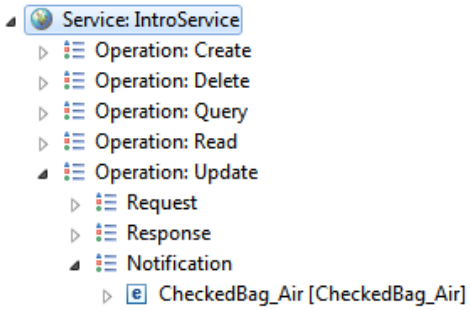
```
<xsd:extension base="ota2msg:OTA2_Response_Payload">
```

The compiler creates a separate directory for services. This directory has examples of the messages and a single XSD schema for each namespace used. The schemas are “Trim” schemas in that all types not needed by the service have been removed.



— Editor Representation

The Editor represents Services a similar fashion to Core and Business Objects. Operations are children of the service and messages are children of their operations. Finally messages are presented like a facet in that they can have properties as children.



Name	Role	Type	D
Operation: Create			
Request			
CheckedBag_Air	Element	CheckedBag_Air	
Response			
CheckedBag_Air	Element	CheckedBag_Air	
Notification			
CheckedBag_AirId...	Element	CheckedBag_Air	
Operation: Delete			
Request			
CheckedBag_Air	Element	CheckedBag_Air	
Response			
CheckedBag_AirId...	Element	CheckedBag_Air	
Notification			
CheckedBag_Air	Element	CheckedBag_Air	
Operation: Query			

