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 Design a Simple MP3 Model Tutorial

Document Purpose:

This document walks you through the features of OpenTravel Model Designer user interface (UI) while you create a simple model of an MP3 collection of songs. The tutorial takes about 15-45 minutes.

Content

Getting Started with Model Designer

Overview

In this guide we will design and create a simple MP3 music model and then compile it to create XML Schemas. The guide begins with an overview of the OTM-DE user interface then walks through a set of tasks that all OTM development projects will follow.

Specifically, this guide outlines the following tasks:

1. Design the Model
2. Create an OTM Library
3. Build OTM Objects
4. Create an Album Service
5. Create Examples
6. Validation
7. Compiling

This introductory project is designed to be completed in 15-45 minutes. The simple music model created during this exercise will then be extended in subsequent, follow-up guides.

Prerequisites

- OTM-DE installed (see installation guide)

Task 1 — Design the Model

Overview: The goal of this task is to create the high-level design of the MP3 Music schema. Typically done with a whiteboard or pen and paper, the process identifies *objects*, self-contained, modular information units.

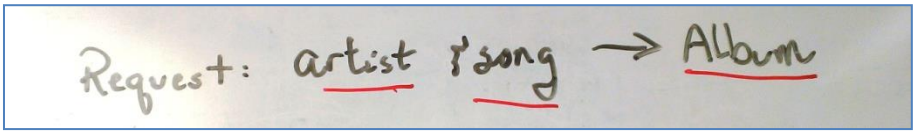
Like a C or Pascal programmer learning object-oriented Java or C++, OTM may require a change in the way you think about XML. Oftentimes service developers begin by thinking about the XML operations they need to implement:

> *"I need a request operation that takes the artist and song returns the album."*

However, the Open Travel Model is an object-oriented model so we need to identify the objects as well as the operations.

1. **Define the services and operations needed.**

   In this project the service has a single operation, albumRequest, which takes an artist and song and returns an album. Below is a white board where the operation was described and then the objects needed were identified and underlined.



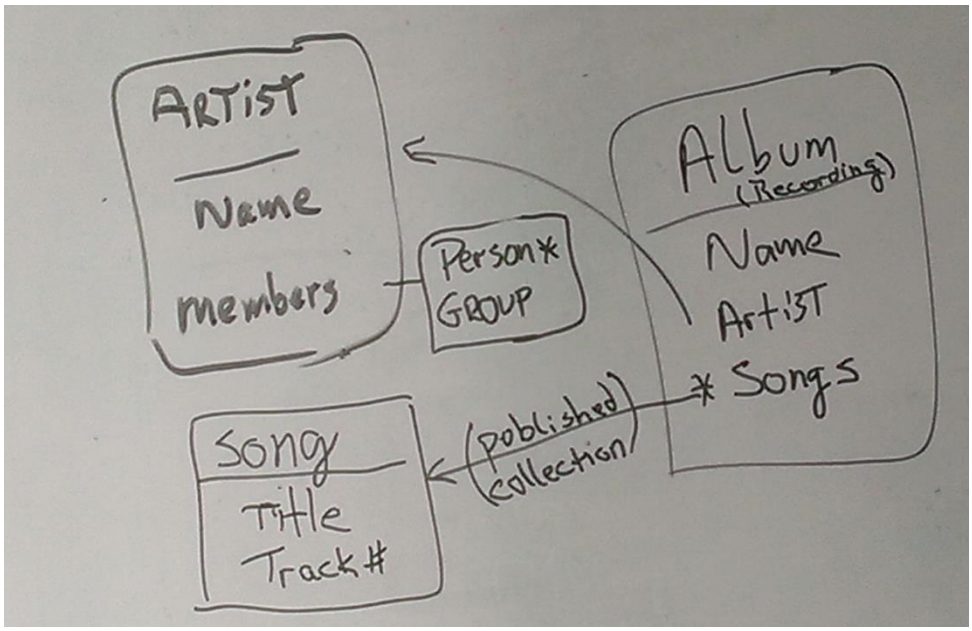
2. **Describe each object in text.**

   Read through the descriptions to make sure that they are consistent. If new objects or alternate names for objects were used, underline them as well.

   1. *Artist* – A person or group who plays a song.
   2. *Song* – A track on an album.
   3. *Album* – A published collection of recordings by an artist.

3. **Diagram objects.**

   Diagramming objects helps to identify the key properties and relationships between objects and helps to clarify when an object needs refactoring because it is not self-defining or sets of its properties are reused in other objects. Diagrams are typically abstract Entity-Relation (ER) or UML diagrams. Diagraming is a good time to work with your subject-matter-experts.



It is most important to capture all the objects and terms underlined in the previous steps than to follow all the rules related to any particular diagramming technique. The above diagram is sufficient for our needs, but does not follow all the rules or either ER or UML in that it uses * to indicate multiples and arrows to capture "type-of" relationships.

Task Summary: working from the statement of the service operation we created a high-level, abstract diagram of objects and their fundamental properties that we will capture in an OpenTravel Model.

Task 2 — Create an OTM Library

Overview: In this task we will create a new library for our music model using the preconfigured namespace, project, and repository. The OTM-DE user interface uses namespaces, projects, and the repository to control vocabulary and provide governance. OTM-DE is preconfigured with a default namespace, project, and local repository over which you have full control.

An Open Travel Model (OTM) contains object and service definitions in *libraries.* OTM Libraries contain:

- Services with Operations and Messages (RQ/RS)
- Complex Objects
- Simple Objects

A library can refer to other libraries. In this project we will use *Built-in* libraries for type definitions. A library is assigned an XML namespace and like XML schemas, multiple libraries can share the same namespace. In OTM-DE, namespaces consist of two parts:

1. **Managed Root** – the beginning of a namespace that is governed by a repository
2. **Extension** – the end of a namespace that is assigned to this library.

Finally, libraries are saved as ".otm" files. These files can be saved in your local file system *or* managed in a repository. Typically, a library will start out in your file system before it is moved to a repository.

1. Choose file system location for your files.

Create a directory in your "My Documents" folder called "OTM Projects." Using a single directory is recommended because every time you open a project or library OTM-DE will begin in the last directory used. Over time this will provide a place to keep three types of OTM-DE files:

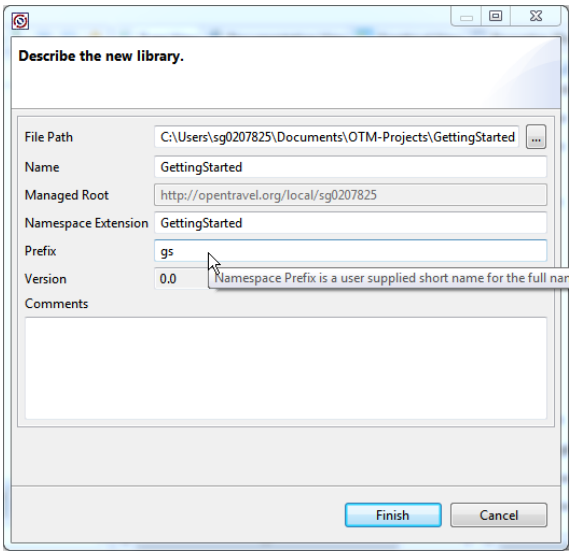
1. Libraries (.otm)
2. Projects (.otp)
3. Compiler output directory (*_CompilerOutput)

Because people manage their local file systems differently, you can choose a different location. Note that you need not make it part of your development archive because the libraries will eventually be managed in a repository.

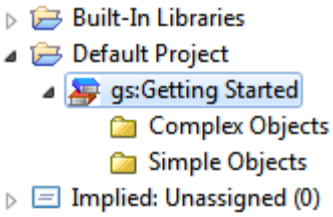
2. Open New Library Wizard

Fill out the four fields:

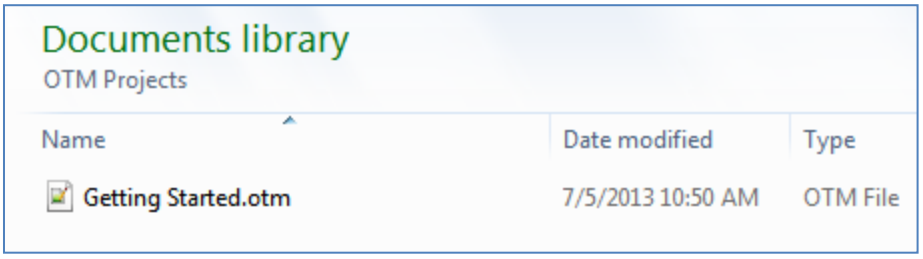
1. **File Path** – use the […] button and select the directory created in task 1. Enter "GettingStarted" as the file name.
2. **Name** – enter "GettingStarted".
3. **Namespace Extension** – enter "GettingStarted" without spaces.
4. **Prefix –** enter "gs".



Task Summary: we created a library to contain the objects and service definitions in our model. The library is in the default project.

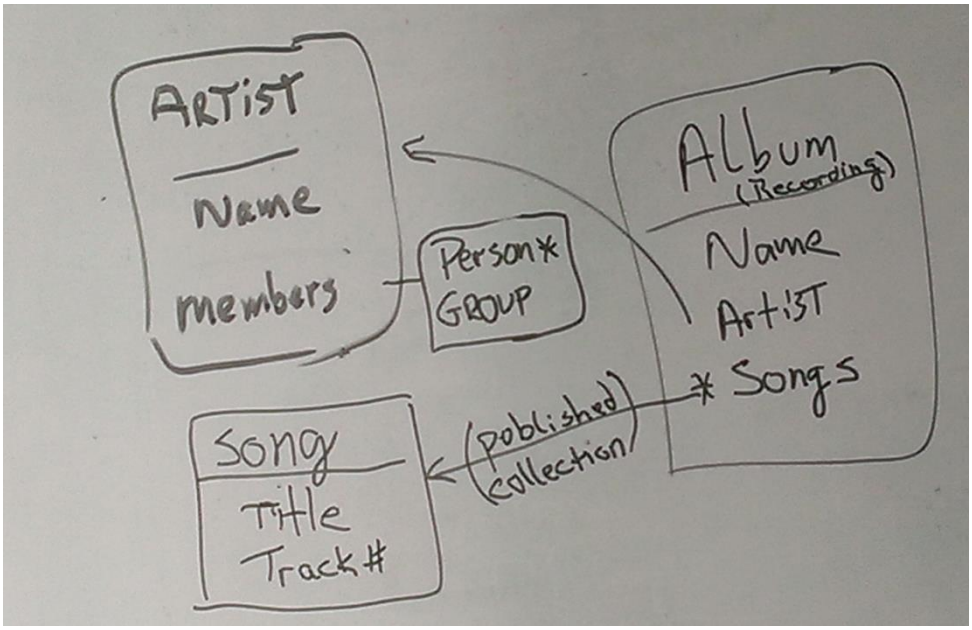


The library file is in your local file system in the "OTM Projects" directory you created.



Task 3 — Build OTM Objects

Overview: In this task you will use the diagram created in first task (copied here for reference) to build OTM objects.

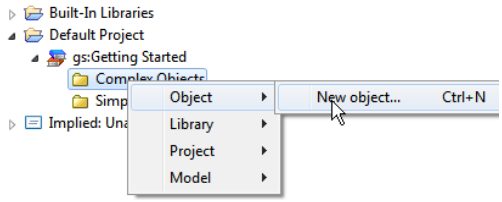


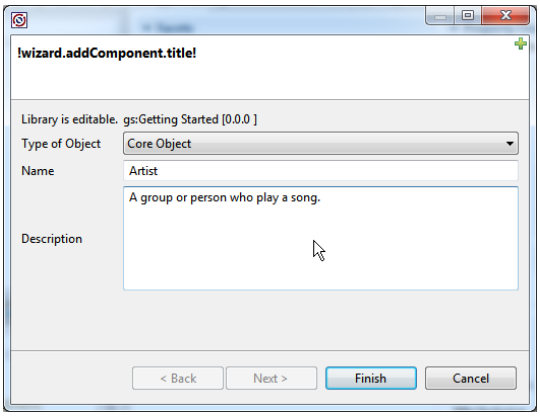
1. Create Artist Object

Artist – A person or group who plays a song.

First, we have to decide which type of object to create. While we could use a *Value With Attributes* for this particular project, we will instead use a *Core Object* in order to make this library extensible for future projects.

1. Right-click on the "Complex Objects" in the "Getting Started" library and select "New object…"

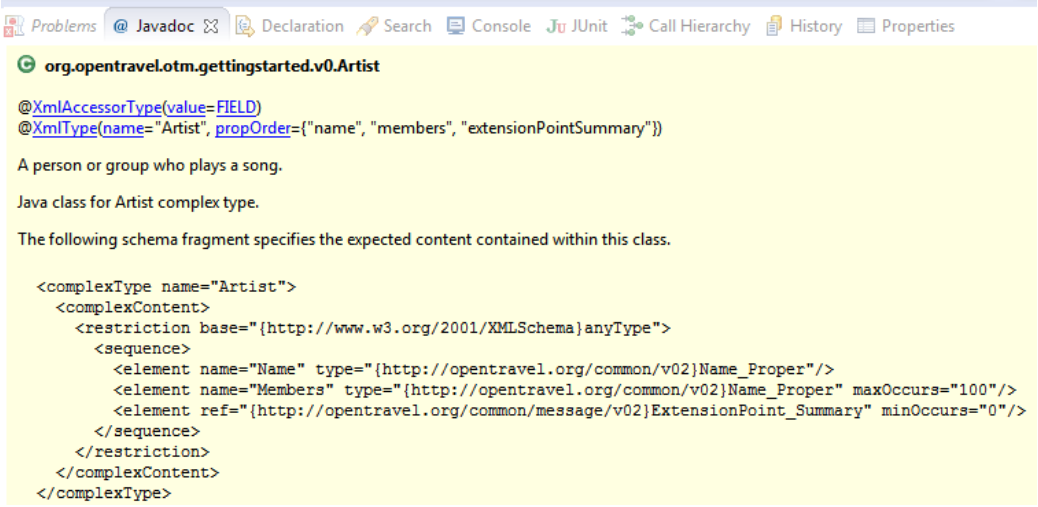




2. Enter the name of the object: "Artist".

3. Enter the description created in the first task: "*A person or group who plays a song.*"

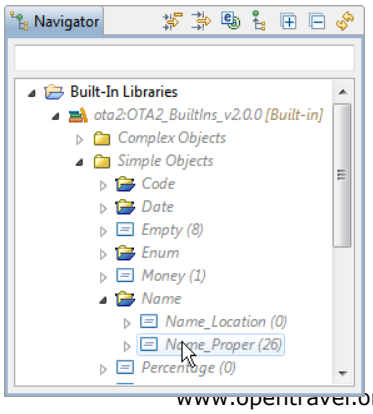
   Note, be sure to always provide good meaningful descriptions because they will become part of the developers interface documentation as in the Javadoc shown here.



2. Add "Name" property.

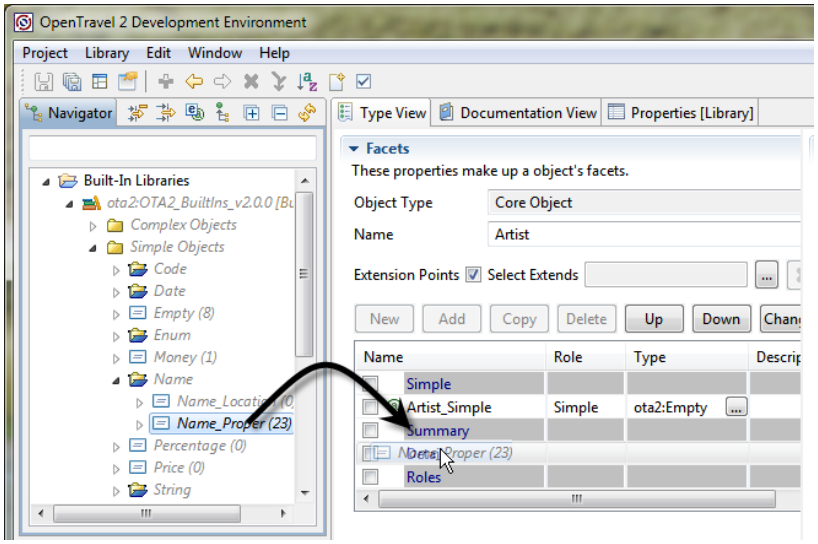
In this step you will use Drag-n-Drop to create an Artist name property. The type of the property will be set to the type dragged out of a Built-In library.

1. Select the "Navigator" tab then find the "Name_Proper" simple object in the OTA2_Builtins library in the "Built-In Libraries" project.

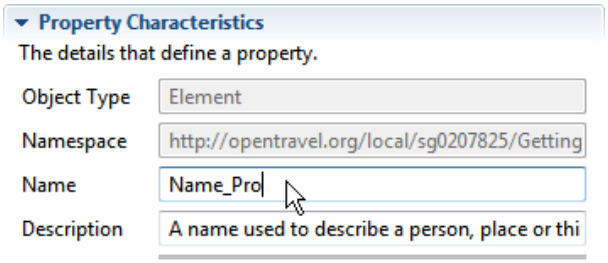


2. If needed, select the newly created Artist core object to display it in the Type View.

3. Drag-n-drop “Name_Proper” onto the Summary facet of the “Artist” core object.

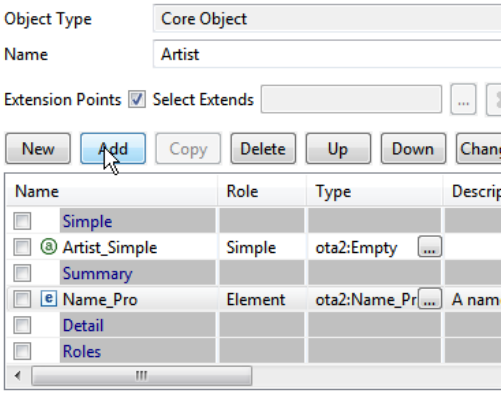


4. Select the property and use the name field in the Properties Characteristics to shorten the name to simply “Name”.

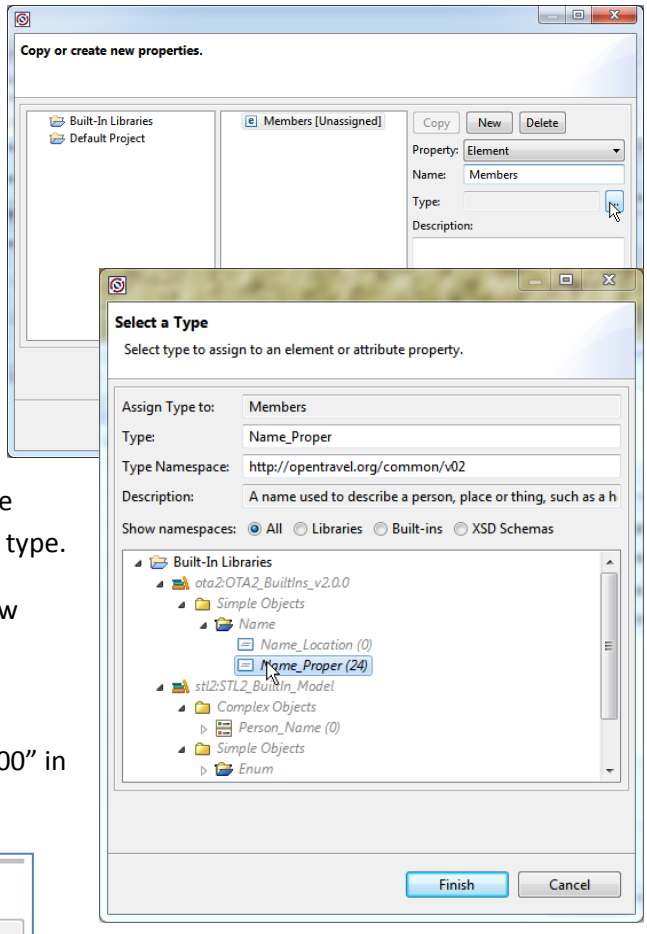


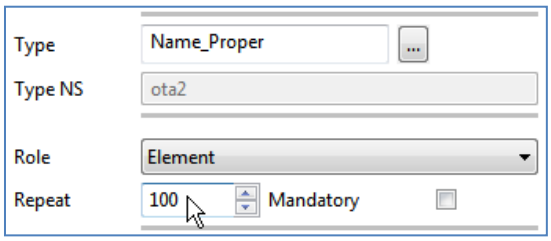
3. Add the Members Property

In this step you will use the “Add Property” wizard to add a “Members” property to your Artist object. Note, the distinction of “person or group” is not required for the service so it will not be captured.



1. Select Artist to be displayed in Type View.
2. Use the “Add” button to launch the New Properties wizard.
3. In the wizard, select “New”.
4. Enter the name: “Members”.
5. Use the “Type” button […] to launch the “Type Selection” wizard.
6. To select the same “Name_Proper” type, begin typing “name” in the type field. Notice how the list is filtered using the name.
7. Select “Name_Proper” from the list. Notice how the menu provides you with the description and namespace of the selected type.
8. Select “Finish” in the type selection and new property wizards. You should now see two properties in the Type View.
9. Select the Members property and enter “100” in the Repeat field.



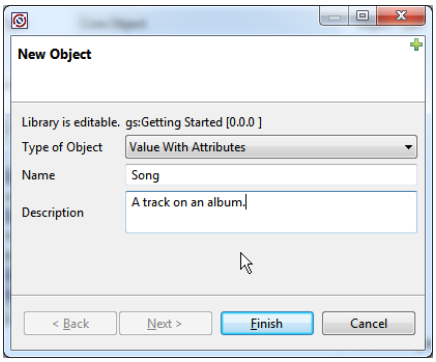


4. Create Song Object

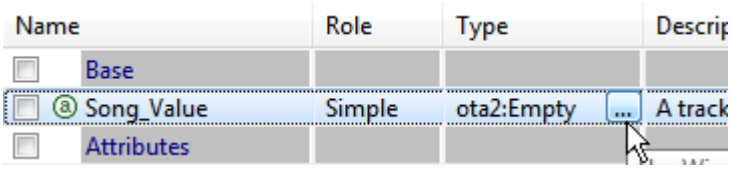
Song – A track on an album.

In this step, we will repeat the techniques utilized previously to create a Song object. Because song only has a name and track number, and because even in future extensions it may only add simple properties such as length, use a *Value With Attributes*.

1. Launch the New Object wizard.

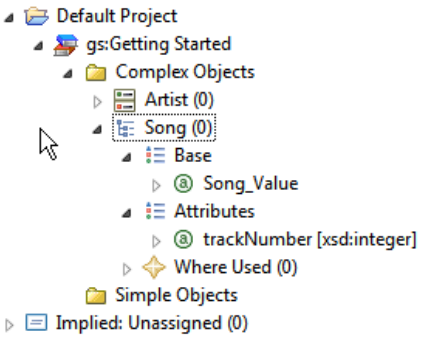


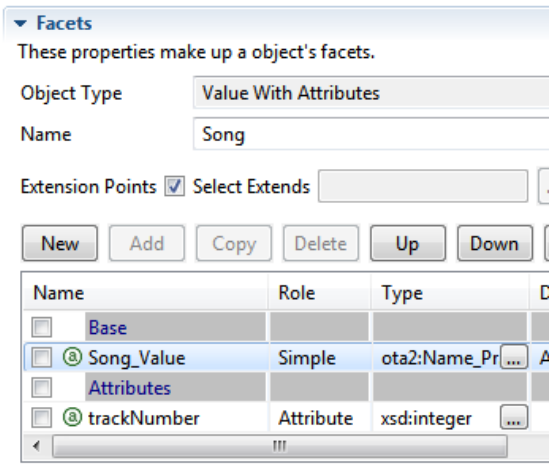
2. Select object type of "Value With Attributes".
3. Enter the name "Song".
4. Enter the description.
5. Launch the type selection wizard for the Song_Value by clicking on the selection button. Select "Name_Proper" as the type.



6. Select the Attributes facet in the type view and use the "Add" button to add the Track Number property. This time, select "integer" from the XML Schema Built-In library. Remember to start typing the type name to filter the list.

Your Song object should look like this in the Navigator and Type views:

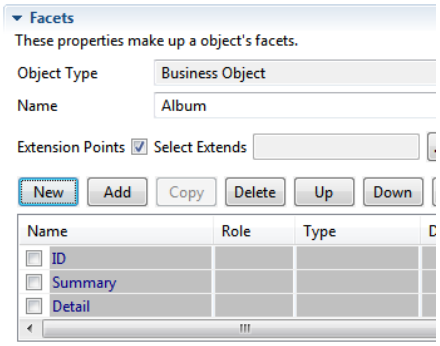




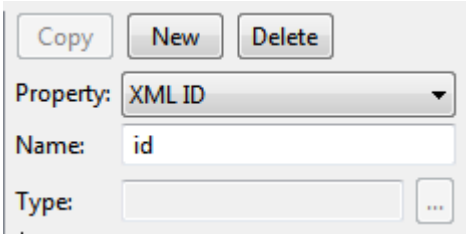
5. Create An Album Object

Album – A published collection of recordings by an artist.

In this step you will use the techniques carried out during the previous steps to complete modeling your objects in the library by adding the Album object. Since Album is a principal object within the model and the target of a service request, we will use a *Business Object*.



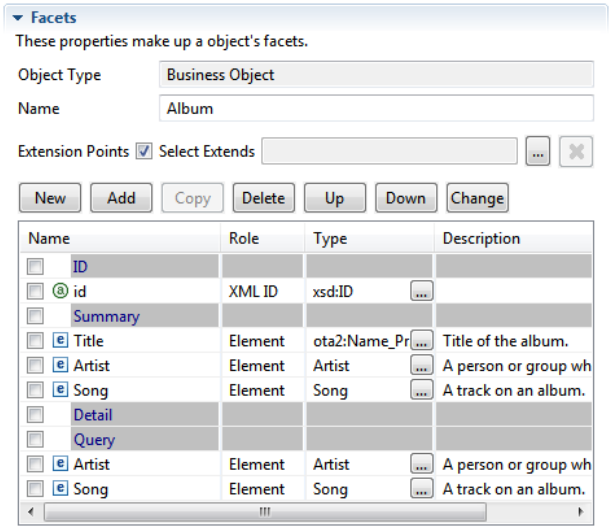
1. Launch the New Object wizard.
2. Select object type of "Business Object**".**
3. Enter the name "Album".
4. Enter the description.
5. Select the ID facet in the type view and use the "Add" button to add an XML ID attribute named "id".



6. Add a Title property with a Name_Proper type from the OTA2 library.
7. Add Artist and Song properties to the Summary facet using Drag-n-Drop.
8. Set the repeat count on Song to 100.

Note the alternate names from the diagram (Recordings and published collection) are not needed for this service so we will not incorporate them into the library at this time.

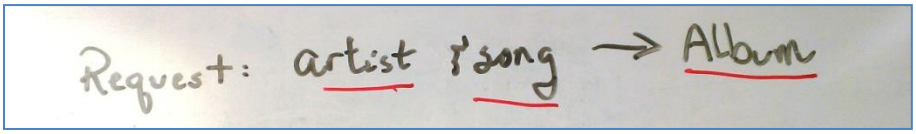
Your completed album object should look like this:



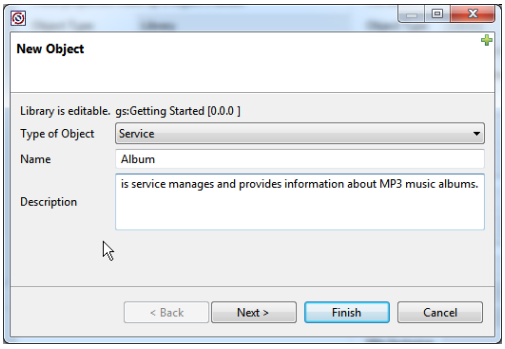
Task Summary: You have used several different wizards and Drag-n-Drop to create three different types of OTM objects based on the model diagramed in the first task.

Task 4 — Create an Album Service

Overview: An OTM service defines how objects are assembled into Request and Response XML messages used by the operations in a service. In this task we will create an Album service that meets the initial requirements.



1. Create a Service Object



1. Use the new object wizard and select "Service" as the object type.
2. Provide a name "Album" and a description.
3. Use the "Finish" button when done.

The "Next" button will be explained in another document.

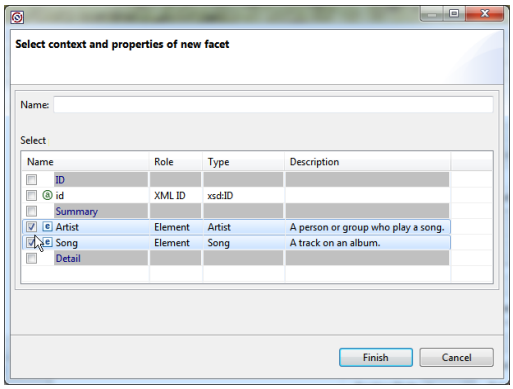
Note, libraries can define only one service so this option will not be presented in the list when the library already has a service.

2. Model the Request

The change in thinking to object-oriented modeling is illustrated by modeling the request. While it may be tempting to say that the request should contain the artist and song, in an OTM model the objects define their own queries for use in requests.

For this service we want to create a "Find" album operation. To model this we want to modify the "Album" business object to add a query facet and add to the query facet the properties needed to *find* the album (song and artist).

3. Add Query facet to the Album business object



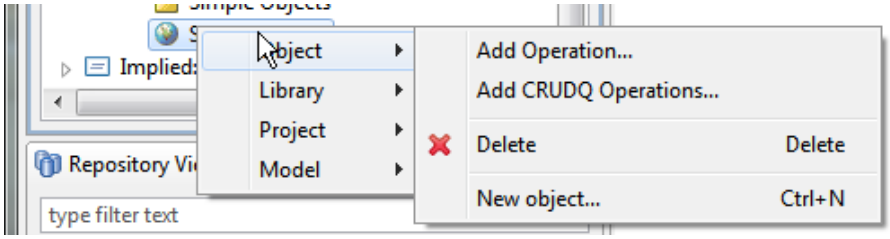
4. Right click on the "Album" business object, select Object-> Add Query Facet …

   A wizard now allows you to select the properties you want to allow this business object to be found by.

5. Select both "Artist" and "Song". Leave the name blank, the product will supply a name.

4. **Add Find operation to the service**

   Right click on the newly created service and select Object->Add Operation… and name the operation “Find”.



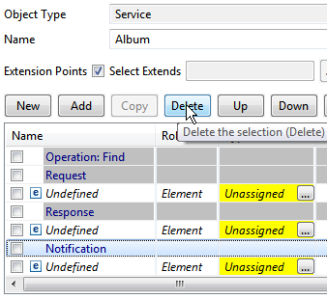
5. **Define Request and Response message properties**

   1. Use Drag-n-drop to set the type of the response to be the “Album” business object.

   2. Expand the navigator view of the album object. Drag the Query facet onto the request property.

6. **Remove Notification facet**

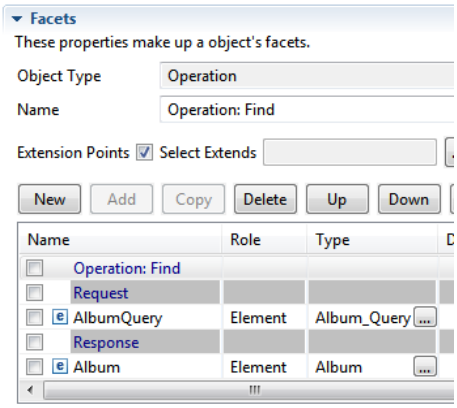
   This service does not support eventing, so we can remove the Notification message. Select the Notification facet and the press the “Delete” button.



Task Summary: Your Service object should now look like the picture.

In this task you added a query facet to the “Album” business object. Not only does this allow the “Album” object to provide a more complete definition, it also allows the song and artist properties in the summary to be mandatory minimizing the amount of optional properties thus simplifying programming the application.

You then used the Album and Album_Query to define the request and response messages used in the Find operation of the Album service.

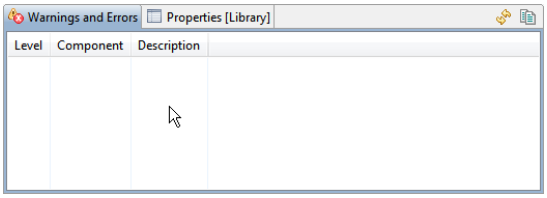


Task 5 — Validation

Overview: Validation checks the model for errors or warnings.

1. **Open Warnings and Errors view**

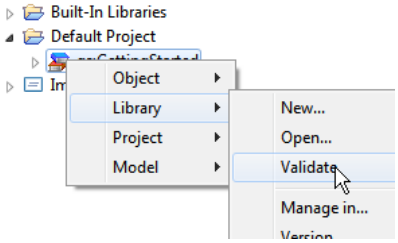
   Select the Warnings and Errors tab to activate the view.



2. **Run Validation**

   Right-click on the GettingStarted library and select Library-> Validate. The findings, if any will be presented in the view.

   If any validation findings are presented, you can double-click on them and the system will open the object that is most likely to have caused the error.

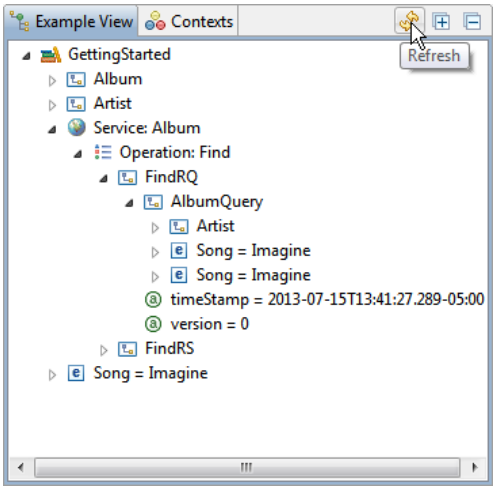


Task 6 — Create Examples

Overview: Two example views can help you check your model to assure it will produce the desired XML.

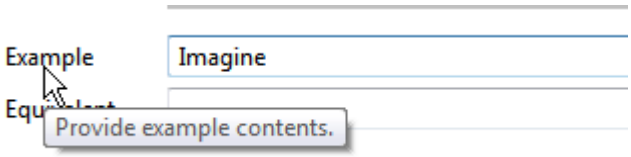
1. **Refresh the Example View**

   Select the refresh arrows in the example view to create examples of your XML objects and services.



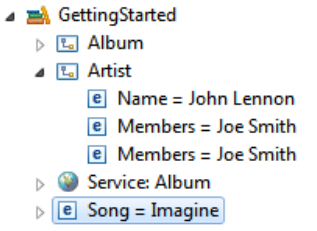
2. **Change example for Song**

   Select the Song value with attributes. Enter "Imagine".



3. **Change example for Artist Name**

   Select Artist. Select Name property. Enter "John Lennon" in the example field.



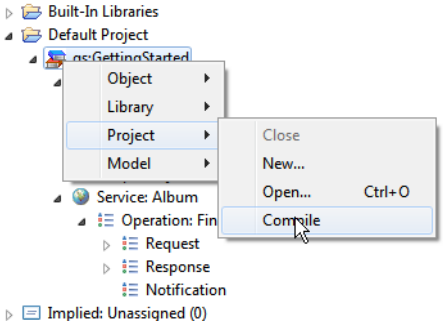
4. **Examine contents of Service**

   Review the contents of the Request and Response to verify they meet the requirements.

Task Summary: You used the example generator and provided real-world example value relevant to the objects. The example was then use to verify the overall service requirements were met.

Task 7 — Compiling

Overview: With a complete and valid model, you are ready to use the compiler to create XML Schemas and a WSDL file that describes your Web service.

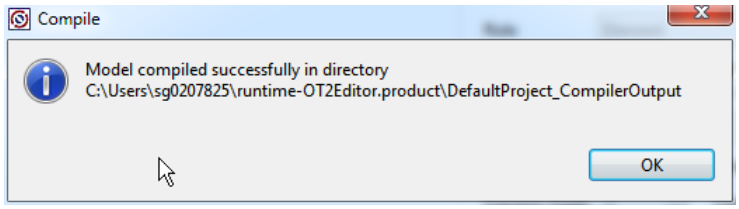


1. **Launch Compiler**

   Right click on the "Getting Started" library and select Project->Compile.

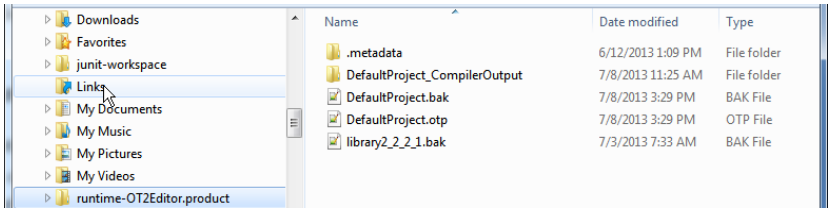
2. **Open the output directory**

   A dialog will be presented that tells you the directory where the output was created.



Your output directory will be where the project file is located. Because we are using the default project, the output is in the install directory.

Note where your output directory is. Find the directory using your file browser before dismissing the compile dialog. The directory could be hidden as by default Windows will hide files and directories beginning with the character "."..



3. **Examine examples in both services and schemas sub-directories**

   Task Summary: With the completion of this task you have completed creating the XML schemas and WSDL files needed to describe a Web service interface.