OTM-DE opentravel2.0

opentravel setting the standard

OTM-DE Namespaces Reference

Document Purpose:

This document describes how both the Designer and the repositories are used to manage namespaces in the OpenTravel Model Development Environment.

Contents

OTM Namespaces

The OTM Repository and Development Environment make extensive use of namespaces. The OTM Designer user interface guides designers in selecting namespaces for their libraries and XSD Schemas, and namespaces are the basis of how the repository controls user access to libraries. This document gives an overview to *namespaces* and how they are used in the OTM Development Environment to implement access controls.

Namespace Overview

In general terms, namespaces make sets of unique names. Said a different way, a namespace is a container for a set of names. The same name can be present in multiple namespaces, but within a namespace the name must be unique.

For people family names are similar to namespaces (e.g. Joe Smith is different from Joe Brown). However family names are not proper namespaces because there is no authority assuring family names are unique—indeed, there is more than one “Joe Smith”.

Authority is an important concept for proper namespaces. There must be an authority that assures names within the namespace are unique and there must be a way to know the identity that specific authority. The OTM Development Environment is the authority for OTM objects and uses XML namespaces to identify the authority. This approach is compatible with XML and most modern programming languages.

XML Namespaces

XML Namespaces when implemented with Namespace aware processors ***can be*** proper namespaces. They define the mechanics of how the authority is to determine if names are unique and which names must be unique. Namespaces in XML[1] defines how to use an URL to identify a namespace authority and further requires namespace aware XML processors to assure that all global element names within the namespace are unique.

To make working with namespaces easier, Namespaces in XML also defines how a *prefix* can be used as a more concise namespace identifier. The standard also defines how to define default namespace to be used when no prefix is present.

In the following XML message, two namespaces are used one with prefix “otm” (http://opentravel.org/OTM/Common/v0) and a default namespace to use when no prefix is present (http://example.com/UserGuides/Patterns/v0).

```
<GetRQ xmlns=”http://example.com/UserGuides/Patterns/v0”
       xmlns:otm="http://opentravel.org/OTM/Common/v0" >
    <GeoData_Query>
        <otm:Type>Airports</otm:Type>
```

[1] Namespaces in XML 1.1 (Second Edition); W3C; 16 August 2006; http://www.w3.org/TR/xml-names11/

```
    </GeoData_Query>
</GetRQ>
```

OTM-DE incorporates all the features of XML namespaces. OTM-DE goes a step further by providing controls around what URL is used for each namespace. Each OTM-DE controlled namespace identifier is guaranteed to be unique relative to all other identifiers managed in the repository. This assures that each namespace:object-name within the repository is unique.

Namespaces in Programming Languages

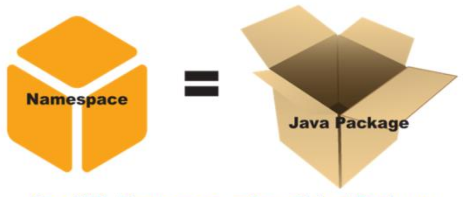
In object oriented programming languages, a namespace (also called name scope) is an abstract container that holds a logical grouping of unique identifiers or symbols (i.e., names).

- In Java, a package declaration identifies to which namespace a class belongs.
- In C++, a namespace is defined with a namespace block.
- In C#/.NET namespaces are declared using explicit *namespace* declarations.

In all cases, the language requires that each namespace/name is unique. Similar to an XML namespace aware processor, the programming and application runtime environments are the authorities that assure all classes are unique within their namespace.

Like XML, programming language namespaces rely upon a namespace identifier. Developers have adopted a convention for their package/namespaces identifiers that also uses the internet naming authority; for example “org.opentravel” as the beginning of a package identifier for an application created by OpenTravel.

Binding tools create programming language classes from XML Schemas. These tools leverage the fact that XML namespace identifiers and package identifiers both use URLs and serve the same purpose. By default these tools directly map the XML namespace to the programming language namespace.



One XML Namespace = One Object Package

The OTM Development Environment controls over XML namespaces therefore also assures class packages created are uniquely identified. This enables application developers to match their software objects to OTM and XML objects, enables software reuse and eases maintenance.

Namespaces and OTM-DE

The OTM Development Environment is designed to provide the authoritative controls over namespace identifiers to assure that each and every object manage within a repository is uniquely identified by its object name and namespace identifier. This gives developers assurance that their programming objects match the data objects found in the messages. And it gives them the assurance they can reuse these programming objects on any message that contains the matching data objects.

To fully leverage the benefits of managed namespaces, OTM-DE uses namespaces to distinguish multiple *versions* of an object, to identify ownership to distinguish OpenTravel defined objects from company specific objects, and to identify and differentiate objects in different business domains. OTM-DE also provides *access controls* based on namespace structures to facilitate team development and implement governance.

Versions

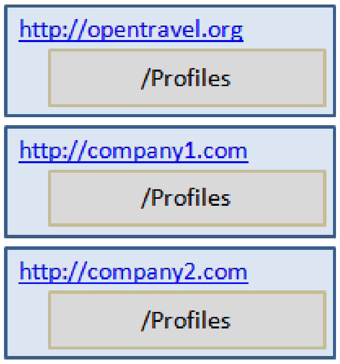
Each namespace managed by OTM-DE ends with a version identifier. This not only assures that each and every version of the object is uniquely identified, it also assures that the version identifier is automatically and consistently applied so that application developers can place their trust in them and take full advantage of the relationships.

Here are examples of namespace identifiers for three different versions a library:

- Major version 1 = http://temp.example.com/test/test/v1
- Minor version 1.1 = http://temp.example.com/test/test/v1_1
- Patch version 1.2.1 = http://temp.example.com/test/test/v1_2_1

Extensions and Ownership

Namespaces are used within OTM-DE to identify the owner of an object as well as extensions to the object. Ownership is defined by the "root namespace" and is assigned by the repository manager. Only namespaces that begin with the root namespace assigned by the repository manager can be saved in that repository.



This allows OTM developers to mix and match content from multiple organizations without the potential of confusion or contention over object names. For example, a designer could create an object that could contain "Profiles" as defined by OpenTravel, *Company1* and *Company2*. OTM-DE will assure that the XML and programming language namespace requirements are met.

Namespace are also used for extensions. Extendable OTM objects automatically have a place added to their XML declaration a place for data from other namespaces to be added. This data **must** be in a different namespace to assure there is no confusion over which data is defined by the OTM object and which is in the extension.

```
 <xsd:element minOccurs="0" ref="ota2msg:ExtensionPoint_Detail"/>

<xsd:complexType name="ExtensionPoint">
  <xsd:sequence>
    <xsd:any minOccurs="0" maxOccurs="unbounded" namespace="##other" processContents="lax"/>
  </xsd:sequence>
```

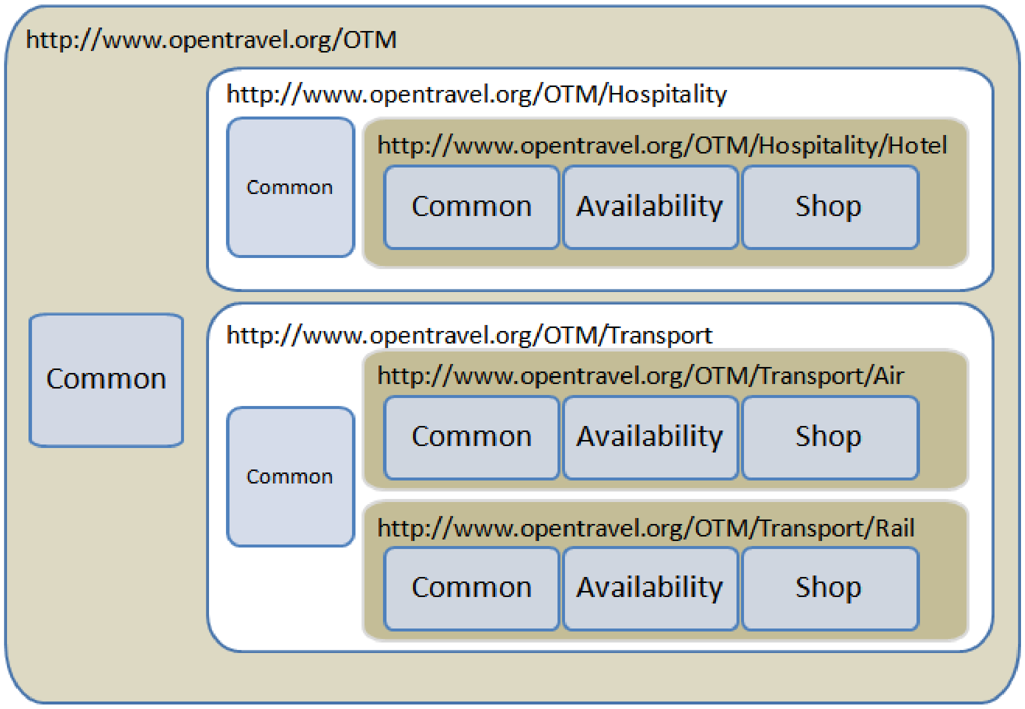
```
</xsd:complexType>
```

Namespace and Business Domains

The repository namespaces are designed to be hierarchical. These hierarchies are designed to permit assignment of namespace permissions to teams and to provide a logical namespace for common, shared content.

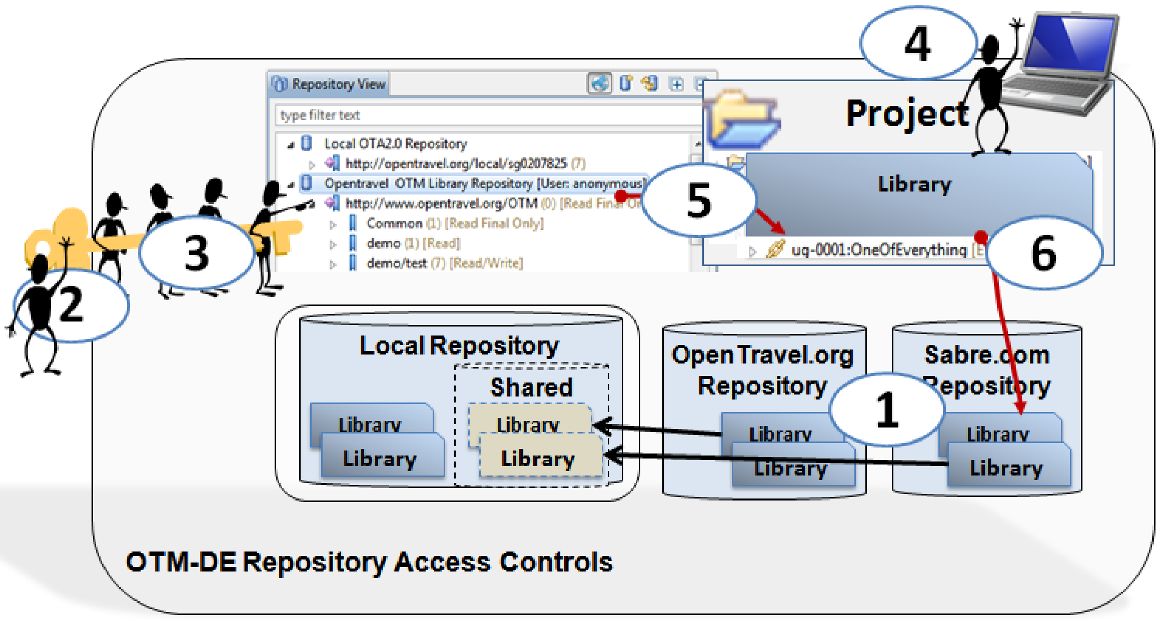
Each repository has one or more *Root Namespace(s)* assigned by the repository administrator. The administrator divides the root namespace into *managed namespace(s).* Library designers can sub-divide the managed namespace by providing an *extension* to their project or library.

Here is an example of how OpenTravel could choose to organize their namespace hierarchy to match their business domains.


http://www.opentravel.org/OTM
http://www.opentravel.org/OTM/Hospitality
Common
http://www.opentravel.org/OTM/Hospitality/Hotel
Common
Availability
Shop
Common
http://www.opentravel.org/OTM/Transport
http://www.opentravel.org/OTM/Transport/Air
Common
Common
Availability
Shop
http://www.opentravel.org/OTM/Transport/Rail
Common
Availability
Shop

Repository Access Controls

Access Controls in the OTM Library Repository are responsible for assuring that OTM Libraries are shared responsibly and to enforce control over the namespaces. OTM Repository access controls are implemented using: users, groups, namespaces, extended-namespaces, libraries and projects. A brief overview of the access control process is described here. See the OTM-DE User Guide – Repositories for more detail.


Repository View
type filter text
Local OTA2.0 Repository
http://opentravel.org/local/sg0207825 (7)
Opentravel OTM Library Repository [User: anonymous]
http://www.opentravel.org/OTM (0) [Read Final Only]
Common (1) [Read Final Only]
demo (1) [Read]
demo/test (7) [Read/Write]
Project
Library
ug-0001:OneOfEverything
Local Repository
Shared
Library
Library
Library
Library
OpenTravel.org Repository
Library
Library
Sabre.com Repository
Library
Library
1
2
3
4
5
6
OTM-DE Repository Access Controls

1. **Repositories** are assigned root-namespaces.
2. **Users** are assigned to groups.
3. **Groups** are granted permissions for namespaces (read-final, read-draft, or write).
4. **Projects** have a namespace chosen from those in the repository.
5. **Libraries** within the project's namespace they can become *editable*.
6. **Libraries** can be *managed* in a repository.

Before a repository will manage a library or allow a library to be committed it will ensure:

1. The library is within the project namespace.
2. The project namespace conforms to the namespaces managed in that repository.
3. The user belongs to a group that has write permission for that namespace.