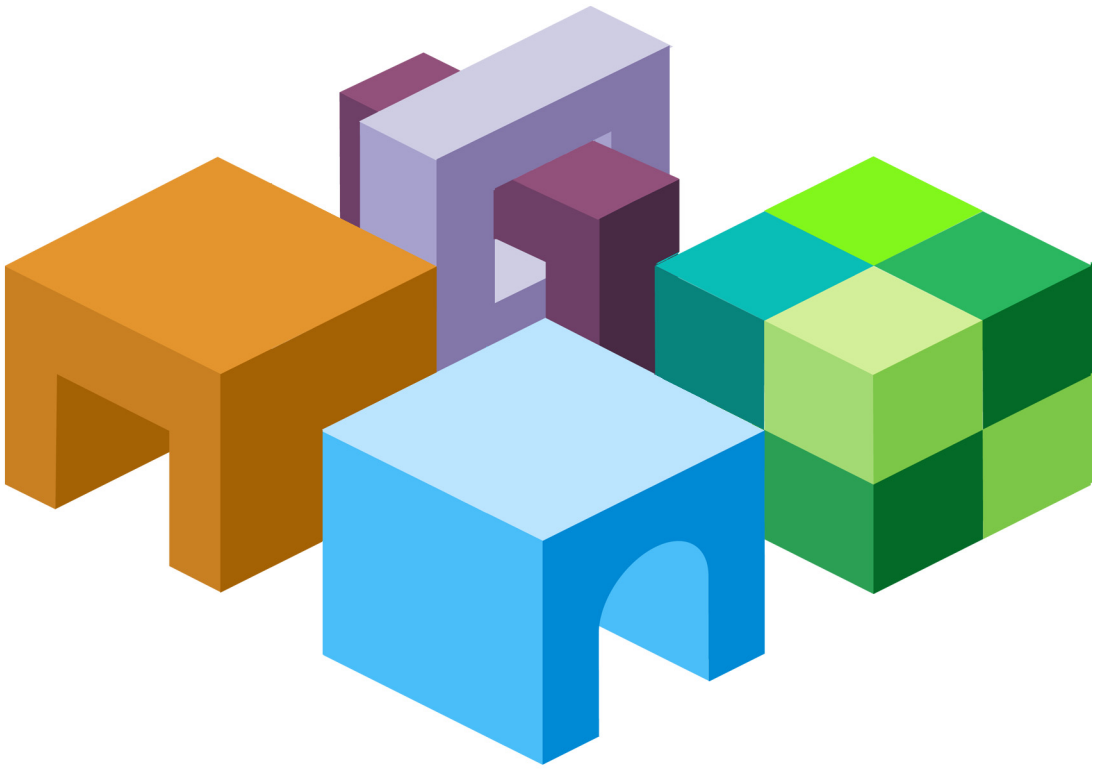


ENTERPRISE PERFORMANCE MANAGEMENT ARCHITECT

RELEASE 9.3.1

WEB SERVICES DEVELOPER'S GUIDE

ORACLE | Hyperion

Performance Management Architect Web Services Developer's Guide, 9.3.1

Authors: Performance Management Architect Information Development

Contents

1 Getting Started

In This Chapter

This guide describes how to use the Oracle's Enterprise Performance Management Architect Web services API, which exposes the Web services layer of Performance Management Architect. Use this API to create client applications that customize Performance Management Architect. Table 1 summarizes the services exposed by the API.

Prerequisites

This guide assumes that you are familiar with the following:

- The .NET Framework
- Web services development and related standards such as SOAP, WSDL, and so on
- Performance Management Architect's features and technical requirements

Most code examples in this guide are written in C#, so some C# familiarity will be helpful.

Technical Reference Documentation

This guide is accompanied by a technical reference help file that provides details on the API's classes and members. The help file is named `bpma_tech_ref.chm`, and is available from the location in which you obtained this document.

About The Web Services API

The Performance Management Architect Web services are .NET Framework Web services. The services are hosted in a virtual directory and run in the IIS process of the application tier (see "Performance Management Architect Architecture" on page 7). The API exposes the Web services listed in Table 1.

Note:

The `.ASMX` files are installed in the `http://<machine name>/hyperion-bpma-server/` directory of the application server. The WSDL-generated proxy class for each service provides a `Url` property that enables you to specify the `.ASMX` file location at run-time. For an example, see Chapter 2, "Sessions Web Service."

Table 1 Exposed Web Services

Web Service	`.ASMX` Filename	Description
Applications	`Applications.asmx`	Creates, deletes, validates, and provides other functionality for Application Views.
Compares	`Compares.asmx`	Compares Application Views, dimensions, and members.
Dimensions	`Dimensions.asmx`	Exposes functionality for working with dimensions.
Folders	`Folders.asmx`	Used to work with Dimension Library folders. For example, you can create folders or move dimensions into folders.
Imports	`Imports.asmx`	Used to import dimensions. For example, you can work with import profiles and perform both flat file and staging area imports.
Jobs	`Jobs.asmx`	Processes and returns status and other information on jobs, such as import and comparison jobs.
Members	`Members.asmx`	Used to work with dimension members.
ObjectRepository	`ObjectRepository.asmx`	Exposes the object repository, which is a container of the Application Views that can be viewed and utilized by users.
Properties	`Properties.asmx`	Used to work with properties of Application Views, dimensions, and members.
Sessions	`Sessions.asmx`	Used to create and close Performance Management Architect sessions. For example, you can use this class to log on.
Utilities	`Utilities.asmx`	Provides helper methods for file transfers.
Libraries	`Libraries.asmx`	Provides access to the application library, which is a container of Application Views.

Service Classes and Data Classes

Each Web service provides classes that fall into the following categories:

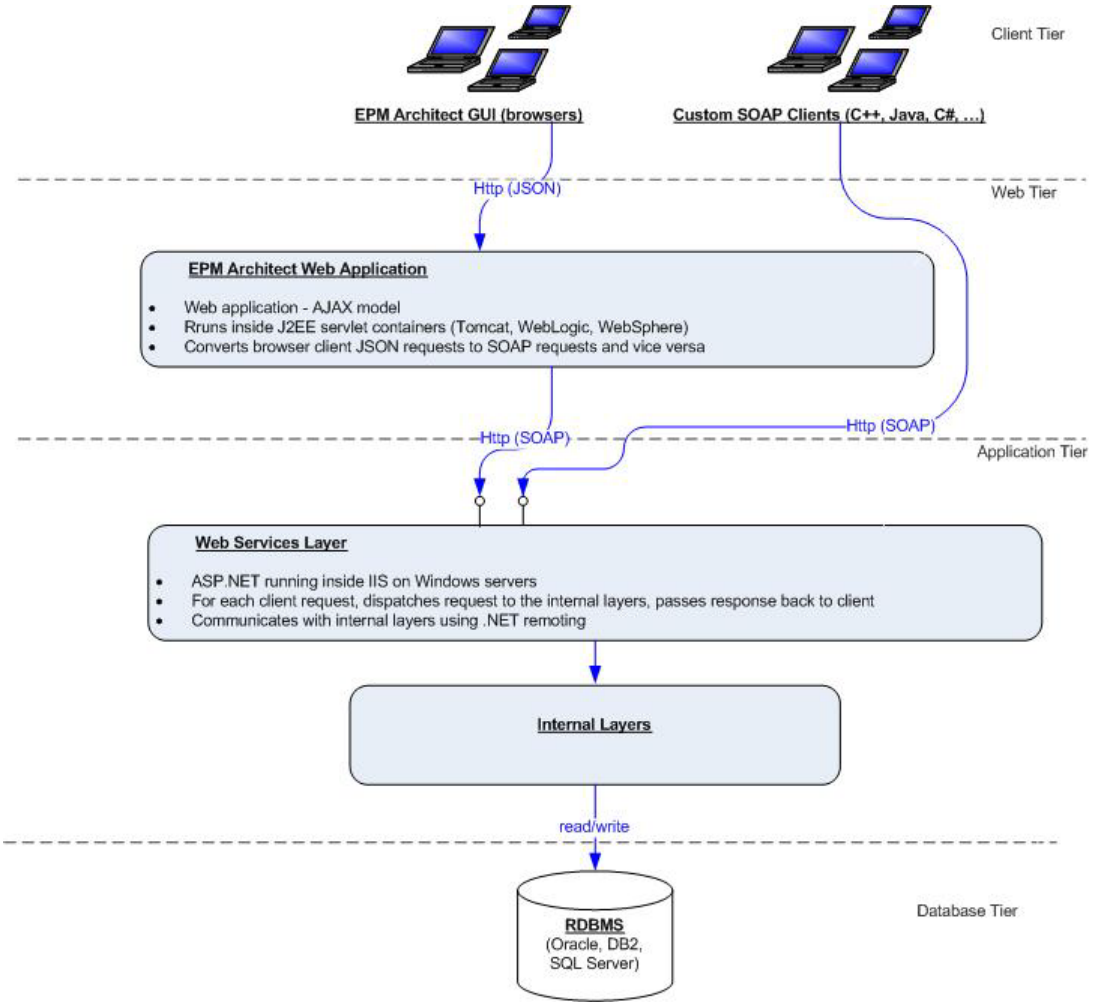
- *Service classes*: Each Web service has one service class. A service class has the same name as the service itself, and implements the operations described in the service's `.ASMX` file. For example, the Applications Web service provides the Applications service class.
- *Data classes*: These classes are provided by the Web services, and primarily are used to transfer data to and from Performance Management Architect, or to access object instances. For

example, the Applications Web service includes the Application data class; an Application object represents an Application View that is being operated upon.

Performance Management Architect Architecture

Performance Management Architect is based on a 4-tier architecture, consisting of client, Web, application, and database tiers; see Figure 1. The Web services API enables you to write SOAP clients that access the Web services layer of the application tier.

Figure 1 Performance Management Architect Multi-Tier Architecture



Locator Objects

There are several data classes that represent “locator” objects. Many of the API's methods take locator objects as parameters; for example, some methods use ApplicationLocator objects to identify Application Views.

Locator objects are a structure for passing multi-column key information. These objects typically contain fields for applicable IDs. For example, an ApplicationLocator object contains fields for the IDs of the Application View, the library that contains the Application View, and the user's session.

2 Sessions Web Service

The Sessions class is used to create a Performance Management Architect session. Most methods require access to a valid session.

To create a session, you must log on. You can pass a username and password with `CreateSession`, or a Oracle's Hyperion® Shared Services token with `CreateSessionFromToken`.

To specify the location of the Web service, set the Sessions class's `Url` property to the following URL, replacing *<machine name>* with the name of the computer hosting Performance Management Architect:

```
http://<machine name>/hyperion-bpma-server/Sessions.asmx
```

`CreateSession` and `CreateSessionFromToken` both return an instance of the SessionInfo data class. SessionInfo provides the `SessionId` field, which you use to access valid sessions. Another useful SessionInfo field is `UserName`, which returns the username associated with the session.

The following C# function logs on, sets a module-level variable with the `SessionId`, and returns the username. The third parameter sets the `Url` property to the location of `Sessions.asmx`.

```
public string SetSession(string sUsername, string sPwd, string sUrl)
{
    //m_SessionsProvider is a Sessions object reference
    m_SessionsProvider.Url = sUrl;
    //SessionsRef is the namespace for the Sessions service's proxy class
    SessionsRef.SessionInfo oSessInfo = m_SessionsProvider.CreateSession(sUsername,
sPwd, string.Empty);
    //m_SessionID is a module-level variable that stores the SessionID
    m_SessionID = oSessInfo.SessionID;
    return oSessInfo.UserName;
}
```

3 Libraries Web Service

The Libraries class is used to access the application library, which is a container of applications. In this release, there is only one supported library, which is known as the master library.

Caution!

The reference documentation lists methods for creating and working with user-defined libraries, but those methods are not supported in this release.

To access the master library, use `GetMasterLibrary`, which returns an instance of the Library data class. An important field provided by the Library class is `LibraryID`, which is used to define LibraryLocator object references.

The following C# function sets a module-level variable to the `LibraryID` of the master library.

```
public void SetMasterLib()
{
    //LibrariesRef is the namespace for the Libraries service's proxy class
    LibrariesRef.Libraries oLibs = new LibrariesRef.Libraries();
    oLibs.Timeout = -1;
    m_MasterLib = oLibs.GetMasterLibrary(m_SessionID);
    //m_LibraryId is a module-level variable that stores the LibraryID
    m_LibraryId = m_MasterLib.LibraryID;
}
```

4 Dimensions Web Service

In This Chapter

The Dimensions class exposes functionality for dimensions. You can use the Dimensions class to perform tasks such as the following:

- Create, copy, and delete dimensions
- Create, copy, and delete dimension associations
- Search for dimensions that match a given property value
- Enumerate dimensions and dimension associations that match various criteria

The following table summarizes commonly-used data classes related to the Dimensions class.

Data Class	Description
Dimension	Represents a dimension, and is identified by the `DimensionID` field. **Note:** A useful Dimension class property is `RootID`, which returns the ID of the dimension's root member.
DimensionAssociation	Represents the definition of a dimension association.
DimensionClass	Represents a dimension type, such as Account or Currency.
Member	Represents a member of a dimension.
DimensionLocator	The locator object used to identify dimensions.

The following topics show how to perform some tasks related to the Dimensions class.

Creating a Dimension

You can create new dimensions with `CreateDimension`, as shown in the following C# example. The example enforces that the passed dimension class is valid by looping through the array of DimensionClass objects returned by `EnumDimensionClasses`.

Note:

A dimension class identifies a dimension's type.

```
//DimensionsRef is the Dimensions service's namespace
public static DimensionsRef.Dimension AddDim(string sName, string sDesc, string sClass,
DimensionsRef.WorkspaceLocator oWsLoc, Guid guidSession)
{
    DimensionsRef.Dimensions oDims = new DimensionsRef.Dimensions();
    DimensionsRef.DimensionClass[] oDimClasses = oDims.EnumDimensionClasses
(guidSession);
    //add dimension only if a valid dimension class was passed
    DimensionsRef.Dimension oRet = new DimensionsRef.Dimension();
    foreach (DimensionsRef.DimensionClass oDimClass in oDimClasses)
    {
        if (oDimClass.Name == sClass)
        {
            oRet = oDims.CreateDimension(oWsLoc, sName, sDesc, sClass);
            break;
        }
    }
    return oRet;
}
```

Enumerating an Application View's Dimensions

The Dimensions service class provides various methods for returning the dimensions contained by an Application View. These include the following methods:

Method	Description
`EnumDimensions`	Returns either all dimensions or only the dimensions of a specified type.
`EnumDimensionsByName`	Returns all dimensions that match the specified dimension names.
`EnumMultiDimensionsByClassName`	Returns a multidimensional array of the dimensions corresponding to the specified dimension classes.

The following C# function returns an array containing the names of an Application View's dimensions. The Application View is specified by the ApplicationLocator parameter.

```
//DimensionsRef is the Dimensions service's namespace
public string[] GetAppDimNames(DimensionsRef.ApplicationLocator oAppLoc)
{
    DimensionsRef.Dimensions oDims = new DimensionsRef.Dimensions();
    DimensionsRef.Dimension[] oDimArr = oDims.EnumDimensions(oAppLoc, new String[0]);
    string[] sNamesArr;
    sNamesArr = new string[oDimArr.Length];
    for (int i = 0; i < oDimArr.Length; i++)
    {
        //set array item to Dimension.Name field
        sNamesArr[i] = oDimArr[i].Name;
    }
```

```
    return sNamesArr;
}
```

Obtaining a DimensionLocator from a Dimension's Name

The following C# function shows how to obtain a DimensionLocator object from a dimension name. The function passes the dimension name to `EnumDimensionsByName`, which returns the `DimensionID` applied to the DimensionLocator object.

```
public DimensionsRef.DimensionLocator GetDimLocator(string sName,
DimensionsRef.ApplicationLocator oAppLoc)
{
    DimensionsRef.Dimensions oDims = new DimensionsRef.Dimensions();
    string[] sArrName = { sName };
    DimensionsRef.Dimension[] oarDims = oDims.EnumDimensionsByName(oAppLoc,sArrName);
    DimensionsRef.DimensionLocator oDimLoc = new DimensionsRef.DimensionLocator();
    foreach (DimensionsRef.Dimension oDim in oarDims)
    {
        if (oDim.Name == sName)
        {
            oDimLoc.ApplicationID = oAppLoc.ApplicationID;
            oDimLoc.LibraryID = oAppLoc.LibraryID;
            oDimLoc.SessionID = oAppLoc.SessionID;
            oDimLoc.DimensionID = oDim.DimensionID;
            break;
        }
    }
    return oDimLoc;
}
```

5 Folders Web Service

The Folders class exposes functionality for folders in the Dimension Library Master View. You can use the Folders class to perform tasks such as the following:

- Create folders
- Rename folders
- Add dimensions to folders
- Delete folders

The following table summarizes commonly-used data classes related to the Folders class.

Data Class	Description
Folder	Represents a folder and is identified by the `FolderID` field.
FolderLocator	The locator object used to identify folders.
FoldersData	Represents a folder, and is identified by the `Folders` field.

The following C# function shows an example of how to create a new folder and add a dimension to the new folder.

```
//FoldersRef is the namespace for the Folders service's proxy class
public FoldersRef.Folder AddFolderAndDim(FoldersRef.FolderLocator
folderLocator, string folderName, string folderDescr, int
dimensionToAddToFolder)
        {
           FoldersRef.Folders oFolders = new FoldersRef.Folders();
           FoldersRef.Folder newFolder =  oFolders.CreateFolder
(folderLocator, folderName, folderDescr);
           folderLocator.FolderID = newFolder.FolderID;
           FoldersRef.Dimension dimensionAddedToFolder =
oFolders.AddDimensionToFolder(folderLocator, dimensionToAddToFolder);
           return newFolder;
        }
```

The Folders service class provides various methods for returning information about what is contained in a folder.

Method	Description
`EnumDimensionsAndFolders`	Returns all dimensions and folders either in a sorted order or filtered.

Method	Description
`EnumDimensionsInFolder`	Returns all dimensions in a folder.
`EnumDimensionsNotInFolder`	Returns a list of all dimensions that are not in a folder.
`EnumFolders`	Returns all subfolders.
`EnumFoldersByDimension`	Returns all folders in which a particular dimension belongs.

6 Applications Web Service

In This Chapter

The Applications class exposes functionality for Application Views. You can use the Applications class to perform tasks such as the following:

- Create, copy, and delete Application Views
- Enumerate all Application Views
- Enumerate the valid application types
- Validate and migrate Application Views
- Add, remove, and enumerate dimensions and dimension members from an Application View

The following table summarizes commonly-used data classes related to the Applications class.

Data Class	Description
Application	Represents an Application View, and is identified by the `ApplicationId` field.
ApplicationClass	Represents an application type, such as Consolidation and Planning.
ApplicationLocator	The locator object used to identify Application Views.
Member	Represents a member of a dimension.

The following topics show how to perform some tasks related to the Applications class.

Obtaining an ApplicationLocator

The following C# function obtains an ApplicationLocator for a given Application View.

```
//ApplicationsRef is the namespace for the Applications web service
public ApplicationsRef.ApplicationLocator GetAppLoc(string sName,
ApplicationsRef.LibraryLocator oMasterLibLoc)
```

```
{
     ApplicationsRef.Applications oApps = new ApplicationsRef.Applications();
     ApplicationsRef.ApplicationLocator oAppLoc = new ApplicationsRef.ApplicationLocator
();
     oAppLoc.SessionID = oMasterLibLoc.SessionID;
     oAppLoc.LibraryID = oMasterLibLoc.LibraryID;
     ApplicationsRef.Application[] oarApps = oApps.EnumApplications(oMasterLibLoc,
null);
     foreach (ApplicationsRef.Application app in oarApps)
     {
         if (app.Name == sName)
         {
            oAppLoc.ApplicationID = app.ApplicationID;
            break;
         }
     }
     return oAppLoc;
}
```

Creating an Application View

To create an Application View, you can use `CreateApplication` or `CopyApplication`. These methods return Application objects. The following C# function creates an Application View. The example uses `EnumApplicationClasses` to validate that the specified application type is valid.

Note:

An Application View created with `CreateApplication` or `CopyApplication` is not displayed to users in the Performance Management Architect GUI unless it is also added to the object repository. See "Adding an Application View to the Object Repository" on page 23. The same principle applies to deleting Application Views; see "Deleting an Application View from the Object Repository" on page 24.

```
//ApplicationsRef is the namespace for the Applications service's proxy class
public ApplicationsRef.Application CreateApp(string sName, string sDesc, string
sAppType, ApplicationsRef.LibraryLocator oMasterLibLoc, Int32 nParAppId)
{
    ApplicationsRef.Applications oApps = new ApplicationsRef.Applications();
    ApplicationsRef.Application oApp = new ApplicationsRef.Application();
    ApplicationsRef.ApplicationClass[] oAppClasses = oApps.EnumApplicationClasses
(oMasterLibLoc.SessionID);
    foreach (ApplicationsRef.ApplicationClass oAppClass in oAppClasses)
    {
        if (oAppClass.Name == sAppType)
        {
            oApp = oApps.CreateApplication(oMasterLibLoc, sName, sDesc, sAppType,
nParAppId);
        }
    }
    return oApp;
}
```

Accessing an Application View from Its Name

Application Views are represented by instances of the Application data class. The following C# function obtains an Application object reference from an Application View name by looping through the Application objects returned by `EnumApplications`.

```
//ApplicationsRef is the namespace for the Applications web service
public ApplicationsRef.Application GetAppFromName(string sName,
ApplicationsRef.LibraryLocator oLibLoc)
{
    ApplicationsRef.Applications oApps = new ApplicationsRef.Applications();
    ApplicationsRef.Application[] oarApps = oApps.EnumApplications(oLibLoc, null);
    ApplicationsRef.Application oAppRet = new ApplicationsRef.Application();
    foreach (ApplicationsRef.Application app in oarApps)
    {
        if (app.Name == sName)
        {
            oAppRet = app;
            break;
        }
    }
    return oAppRet;
}
```

Working with an Application View's Dimension Members

You can use `IncludeMember` and `ExcludeMember` to manipulate the members contained by a dimension in an Application View. You also can enumerate an Application view dimension's members with `EnumIncludedMembers` and `EnumExcludedMembers`.

The following C# function returns the excluded members of an application and dimension.

```
//ApplicationsRef,DimensionsRef are namespaces for the Applications,Dimensions services
public string[] GetExcludedMemsOfDim(string sDimName, DimensionsRef.ApplicationLocator
oAppLoc)
{
    ApplicationsRef.Applications oApps = new ApplicationsRef.Applications();
    ApplicationsRef.DimensionLocator oDimLoc = new ApplicationsRef.DimensionLocator();
    DimensionsRef.Dimensions oDims = new DimensionsRef.Dimensions();
    oDimLoc.LibraryID = oAppLoc.LibraryID;
    oDimLoc.SessionID = oAppLoc.SessionID;
    //get an array of the Application View dimensions
    DimensionsRef.Dimension[] oArrDim = oDims.EnumDimensions(oAppLoc, new String[0]);
    oDimLoc.ApplicationID = oAppLoc.ApplicationID;
    //set the locator to the dimension
    foreach (DimensionsRef.Dimension dim in oArrDim)
    {
        if (dim.Name == sDimName)
        {
            oDimLoc.DimensionID = dim.DimensionID;
            break;
        }
    }
    ApplicationsRef.QueryResultSet oRsSet = oApps.EnumExcludedMembers(oDimLoc);
```

```
    string[] sRet = new string[oRsSet.Members.Length];
    for (int i = 0; i < oRsSet.Members.Length; i++)
    {
        sRet[i] = oRsSet.Members[i].Name;

    }
    return sRet;
}
```

7 ObjectRepository Web Service

In This Chapter

The ObjectRepository service class exposes the object repository, which is a container of the Application Views that can be viewed and utilized by users. When an Application View is created with the Applications class, it is not displayed in the Performance Management Architect GUI. To make a programmatically-created Application View visible, you must add it to the object repository.

Note:

If you delete an Application View with `Applications.DeleteApplication`, and the application is in the object repository, you should also delete it from the object repository with `DeleteObject`. Otherwise, the icon for the deleted Application View would display in the Oracle's Enterprise Performance Management Architect GUI.

Objects in the repository are represented by instances of the RepositoryObject data class.

The following topics show how to perform some tasks related to the ObjectRepository class.

Adding an Application View to the Object Repository

The following C# method adds an Application View to the object repository with `SaveObject`.

```
public void AddAppToObjRepos(string sAppName, string sAppId, string sAppType, string
sUser, string sTimeStamp)
{
    //ObjectRepositoryRef is the namespace for the ObjectRepository service
    ObjectRepositoryRef.ObjectRepository oObjRep = new
ObjectRepositoryRef.ObjectRepository();
    ObjectRepositoryRef.RepositoryObject oRepObj = new
ObjectRepositoryRef.RepositoryObject();
    //m_MasterLib is an object reference to the master library
    string sMasterLibID = m_MasterLib.LibraryID.ToString();
    //Object ID is library ID, underscore, App View ID
    oRepObj.ObjectID = sMasterLibID + "_" + sAppId;
```

```
    //object status = non-deployed
    oRepObj.ObjectStatus = "1";
    oRepObj.ObjectType = sAppType;
    oRepObj.ObjectValue = "<object id=\"" + sMasterLibID + "_" + sAppId + "\" name=\"" +
sAppName + "\" folderId=\"root\" type=\"" + sAppType + "\" owner=\"" + sUser + "\"
created=\"" + sTimeStamp + "\" status=\"1\" modified=\"" + sTimeStamp +
"\"><description /><applicationView><dimensionServerLibraryID>" + sMasterLibID + "</
dimensionServerLibraryID><dimensionServerApplicationID>" + sAppId + "</
dimensionServerApplicationID><viewCreationDate transId=\"-1\"><dateTime>" + sTimeStamp +
" </dateTime></viewCreationDate></applicationView></object>";
    oObjRep.SaveObject(m_SessionID, oRepObj);
}
```

Deleting an Application View from the Object Repository

The following C# method deletes an Application View. The method also checks the object repository for the Application View, and deletes it if found.

`Applications.DeleteApplication` deletes the Application View; `EnumObjects` and `DeleteObject` are used to delete the Application View from the object repository.

```
//ApplicationsRef is the namespace for the Applications service
public void DeleteApp(ApplicationsRef.ApplicationLocator oAppLoc)
{
    ApplicationsRef.Applications oApps = new ApplicationsRef.Applications();
    //ObjectRepositoryRef is the namespace for the ObjectRepository service
    ObjectRepositoryRef.ObjectRepository oObjRep = new
ObjectRepositoryRef.ObjectRepository();
     oApps.DeleteApplication(oAppLoc);
    //enum all RepositoryObjects
    ObjectRepositoryRef.RepositoryObject[] oRepObjs = oObjRep.EnumObjects
(m_SessionID,-1, -1, string.Empty);
    string sRepObjId = m_MasterLib.LibraryID.ToString() + "_" +
oAppLoc.ApplicationID.ToString();
    foreach (ObjectRepositoryRef.RepositoryObject reposObj in oRepObjs)
    {
        //if the App View is in the object repository, delete from it
        if (reposObj.ObjectID == sRepObjId)
        {
            oObjRep.DeleteObject(oAppLoc.SessionID, sRepObjId);
            break;
        }
    }
}
```

8 Members Web Service

In This Chapter

The Members class exposes functionality for dimension members. You can use the Members class to perform tasks such as the following:

- Create, copy, and delete members
- Insert, remove, and exclude members
- Search for members by name or property value
- Enumerate members that match various criteria

The following table summarizes commonly-used data classes related to the Members class.

Data Class	Description
Member	Represents a dimension member, and is identified by the `MemberID` and `ParentID` fields.
MemberLocator	The locator object used to identify members.
QueryResultSet	Stores the results of a member query. The `Members` field returns an array of QueryMember objects.
QueryMember	Represents one member returned by a member query.

The following topics show how to perform some tasks related to the Members class.

Creating a Member

You can create a new dimension with `CreateMember`, as shown in the following C# example. The example creates a member under the specified dimension's root member.

```
//MembersRef and DimensionsRef are namespaces for the Members/Dimensions services
public MembersRef.Member AddMemberToDimRoot(DimensionsRef.DimensionLocator oDimLoc,
string sName, string sDesc, string sClass)
{
    MembersRef.Members oMems = new MembersRef.Members();
    MembersRef.MemberLocator oParLoc = new MembersRef.MemberLocator();
```

```
    DimensionsRef.Dimensions oDims = new DimensionsRef.Dimensions();
    DimensionsRef.Dimension oDim = oDims.GetDimension(oDimLoc);
    oParLoc.SessionID = oDimLoc.SessionID;
    oParLoc.ApplicationID = oDimLoc.ApplicationID;
    oParLoc.LibraryID = oDimLoc.LibraryID;
    oParLoc.DimensionID = oDimLoc.DimensionID;
    oParLoc.MemberID = oDim.Root.MemberID;
    oParLoc.ParentID = oDim.Root.ParentID;
    MembersRef.Member oMem = oMems.CreateMember(oParLoc, sName, sDesc, sClass);
    return oMem;
}
```

Searching for a Member by Name

The following C# function shows how to search for a member with `SearchMembersByName`.

```
//MembersRef is the namespace for the Members Web service proxy class
public MembersRef.Member GetMemFromName(DimensionsRef.DimensionLocator oDimLoc, string
sName)
{
    MembersRef.Members oMems = new MembersRef.Members();
    MembersRef.Member oMem = new MembersRef.Member();
    MembersRef.MemberLocator oMemLoc = new MembersRef.MemberLocator();
    oMemLoc.SessionID = oDimLoc.SessionID;
    oMemLoc.ApplicationID = oDimLoc.ApplicationID;
    oMemLoc.LibraryID = oDimLoc.LibraryID;
    oMemLoc.DimensionID = oDimLoc.DimensionID;
    oMemLoc.MemberID = -1;
    oMemLoc.ParentID = -1;
    MembersRef.QueryResultSet oQueryResultSet = oMems.SearchMembersByName(oMemLoc,
sName, 100, null, null);
    for (int i = 0; i < oQueryResultSet.Members.Length; i++)
    {
        if (oQueryResultSet.Members[i].Name == sName)
        {
            oMemLoc.MemberID = oQueryResultSet.Members[i].MemberID;
            oMemLoc.ParentID = oQueryResultSet.Members[i].ParentID;
            oMem = oMems.GetMember(oMemLoc);
            break;
        }
    }
    return oMem;
}
```

Enumerating a Member's Ancestors

The Members service class provides various methods for enumerating members that meet various criteria. The following C# function returns an array containing the names of a given member's ancestors.

```
//MembersRef is a namespace for the Members Web service
public string[] GetAncestorNames(MembersRef.MemberLocator oMemLoc)
{
```

```
    MembersRef.Members oMems = new MembersRef.Members();
    MembersRef.Member[] oMem = oMems.EnumAncestorMembers(oMemLoc, false);
    string[] sRet = new string[oMem.Length];
    for(int i = 0; i < oMem.Length; i++)
    {
        sRet[i] = oMem[i].Name;
    }
    return sRet;
}
```

9 Properties Web Service

In This Chapter

The Properties class exposes functionality for the properties, such as those of dimension members and Application Views. You can use the Properties class to perform tasks such as the following:

- Obtain and update the value of a member's property
- Enumerate property categories
- Enumerate property definitions
- Obtain the definition of a given property

The following table summarizes commonly-used data classes related to the Properties class.

Data Class	Description
PropertyCategory	Represents a property category, such as Consolidation or Planning.
PropertyDefinition	Represents a definition of a property. A PropertyDefinition object includes fields that define a property's range of valid values, such as: ● `DataType` ● `AllowedValues` ● `MaximumLength` A PropertyDefinition object also provides the `PropertyID` field, which uniquely identifies each property.
PropertyLevel	Represents the type of item to which a property applies. For example, there are properties for dimension members, dimensions, and Application views.
PropertyValue	Represents the value of a property.
PropertyLocator	The locator object used to identify properties.

The following topics show how to perform some tasks related to the Properties class.

Obtaining a PropertyDefinition Object

The following C# function uses `PropertyDefinitionByName` to return the PropertyDefinition object that represents a given property.

```
//PropertiesRef is the namespace for the Properties service's proxy class
public PropertiesRef.PropertyDefinition GetPropertyDef(string sName)
{
    PropertiesRef.Properties oProps = new PropertiesRef.Properties();
    //m_SessionID stores SessionInfo.SessionID
    PropertiesRef.PropertyDefinition oPropDef = oProps.PropertyDefinitionByName
(m_SessionID,sName);
    return oPropDef;
}
```

Obtaining a PropertyValue Object

`EnumPropertyValues` returns an array of PropertyValue objects. The following C# example obtains a PropertyValue object for a given dimension member property.

```
//PropertiesRef is the namespace for the Properties Web service
public PropertiesRef.PropertyValue GetMemVal(Int32 nAppId, Int32 nDimId, Int32
nMemberId, Int32 nParentId, string sName)
{
    PropertiesRef.Properties oProps = new PropertiesRef.Properties();
    PropertiesRef.PropertyLocator oPropLoc = new PropertiesRef.PropertyLocator();
    //for PropertyLocator, use variables that store IDs at the module level
    oPropLoc.SessionID = m_SessionID;
    oPropLoc.LibraryID = m_LibraryId;
    //set other IDs from parameters
    oPropLoc.ApplicationID = nAppId;
    oPropLoc.DimensionID = nDimId;
    oPropLoc.MemberID = nMemberId;
    oPropLoc.ParentID = nParentId;
    string[] sNameArr = { sName };
    PropertiesRef.PropertyValue[] oPropVals = oProps.EnumPropertyValues(oPropLoc,
sNameArr);
    return oPropVals[0];
}
```

Getting Properties for a Dimension Class

`EnumPropertyDefinitionsByDimensionClass` returns an array of PropertyDefinition objects that represent the properties applicable to a dimension class. The following C# example shows how to use `PropertyDefinition.CategoryNames` to return the names of the properties for a dimension class within a given category.

```
public string[] GetPropNamesDimCategory(string sDimClass, string sCategory)
{
    //PropertiesRef is the namespace for the Properties Web service
    PropertiesRef.Properties oProps = new PropertiesRef.Properties();
    PropertiesRef.PropertyLevel oPropLevel = new PropertiesRef.PropertyLevel();
```

```
    //m_SessionID stores SessionInfo.SessionID
    PropertiesRef.PropertyDefinition[] oPropDefs =
oProps.EnumPropertyDefinitionsByDimensionClass(m_SessionID, sDimClass, oPropLevel,
null);
    string sbuild = null;
    for (int i = 0; i < oPropDefs.Length; i++)
    {
        string[] sCats = oPropDefs[i].CategoryNames;
        for (int j = 0; j < sCats.Length; j++)
        {
            if (sCats[j] == sCategory)
            {
                sbuild = sbuild + oPropDefs[i].Name;
                if (i < oPropDefs.GetUpperBound(0))
                {
                    sbuild = sbuild + ",";
                }
            }
        }
    }
    char[] cDelimiter = {','};
    return sbuild.Split(cDelimiter);
}
```

10 Jobs Web Service

In This Chapter

The Jobs class exposes the ability to process and monitor jobs such as imports and comparisons. You can use the Jobs class to perform tasks such as the following:

- Submit a new job
- Check on the status and progress of running jobs
- Delete jobs
- Enumerate jobs

The following table summarizes commonly-used data classes related to the Jobs class.

Data Class	Description
JobInformation	Represents job information such as the job's ID, user, progress, and so on.
ServerJob	Represents a given job. ServerJob objects are returned by classes other than Jobs that launch jobs, such as `Imports.StartImportJob`.
JobAttachment	Represents an attachment for a job. A job can have one or more attachments.
JobFilter	Represents criteria by which to filter jobs.
JobData	Represents the jobs that match given filtering criteria.

In addition, the JobStatus enumeration provides constants that represent the valid job statuses.

Tip:

You can return a job's status with `GetJobStatus`.

The following topics show how to perform some tasks related to the Jobs class.

Processing a Job

A common task is to monitor the status of a ServerJob object until it is completed, as shown in the following C# function.

Note:

For import jobs, you can also return information on import results; see "Processing an Import Job" on page 40.

```
public bool ProcessJob(Int32 nJobId)
{
    //JobsRef is the namespace for the Jobs service's proxy class
    JobsRef.Jobs oJobs = new JobsRef.Jobs();
    JobsRef.JobStatus oJobStatus = new JobsRef.JobStatus();
    bool bRet = false;
    do
    {
        Thread.Sleep(2000);
        //m_SessionID stores SessionInfo.SessionID
        oJobStatus = oJobs.GetJobStatus(m_SessionID, nJobId);
        if (oJobStatus == JobsRef.JobStatus.Completed)
        {
            bRet = true;
        }
    } while ((oJobStatus != JobsRef.JobStatus.Completed) && (oJobStatus !=
JobsRef.JobStatus.Aborted));
    return bRet;
}
```

Filtering Jobs

To filter a job, you set the filtering properties of a JobFilter object and pass it to `EnumJobs`, which returns a JobData object. The `JobData.Jobs` property returns an array of JobInformation objects representing the jobs that match the filtering criteria. The following C# example returns JobInformation objects that represent jobs of a given type for the currently-connected user.

```
//JobsRef is the namespace for the Jobs service's proxy class
public JobsRef.JobInformation[] FilterUserJobsByType(string[] sTypes)
{
    JobsRef.Jobs oJobs = new JobsRef.Jobs();
    JobsRef.JobFilter oFilter = new JobsRef.JobFilter();
    oFilter.JobType = sTypes;
    //-1 for JobID allows all jobs to be filtered through
    oFilter.JobID = -1;
    oFilter.FromSubmittedTime = DateTime.Today;
    oFilter.ToSubmittedTime = DateTime.Today;
    //m_SessionID stores SessionInfo.SessionID (current user)
    JobsRef.JobInformation[] oJobInfos = oJobs.EnumJobs(m_SessionID,oFilter).Jobs;
    return oJobInfos;
}
```

11 Imports Web Service

In This Chapter

The Imports class exposes functionality for importing. You can use the Imports class to perform tasks such as the following:

- Create and delete import profiles
- Upload flat files
- Work with interface tables
- Start import jobs

The following table summarizes commonly-used data classes related to the Imports class.

Data Class	Description
ImportDimensionMapping	Specifies how dimensions in the import source will be imported. The ImportDimensionMapping class's `SourceDimension` and `TargetDimension` fields use the Dimension data class.
ImportPropertyMapping	Specifies how properties in the import source will be imported. The ImportPropertyMapping class's `SourceProperty` and `TargetProperty` fields use the PropertyDefinitions data class.
ServerJob	Represents an import job, and is returned by methods that start imports. You can monitor the progress of the job with the Jobs class; see Chapter 10, "Jobs Web Service."
ImportTypeInfo	Represents the type of import, such as flat file or interface table.
ImportProfile	Represents an import profile.
ImportFlatFileProfile	Exposes flat file profile properties such as the delimiter and whether to trim leading and trailing spaces.
ImportProfileInfo	Exposes profile information such as the profile's name and `ImportID`.

Data Class	Description
ImportResultSet	Contains the results of the import job. Results include data such as counts of the dimensions and properties created.
JobAttachmentBatchLocator	Points to the attachment for a given import job, and is used to obtain import results.

Note:

The Utilities service class provides helper methods related to importing. See Chapter 12, "Utilities Web Service."

The following topics show how to perform some tasks related to the Imports class.

Creating an Import Profile

You can create an import profile with `CreateImportProfile`. The following C# function creates an import profile for a flat file import.

```
//ImportsRef is the Imports service namespace
public ImportsRef.ImportProfile CreateFlatProfile(string sName, string sDesc, string
sFile)
{
    ImportsRef.Imports oImports = new ImportsRef.Imports();
    //m_SessionID stores SessionInfo.SessionID
    ImportsRef.ImportProfile oImpProfile = oImports.CreateImportProfile(m_SessionID,
sName, sDesc, ImportsRef.ImportType.FlatFile, sFile);
    return oImpProfile;
}
```

Accessing an Import Profile

You can access an import profile by looping through the array of ImportProfileInfo objects returned by `EnumImportProfileInfos`. An ImportProfileInfo object includes the `Name` and `ImportID` fields. You can then obtain an ImportProfile object reference by passing the `ImportID` to `GetImportProfile`, as shown in the following C# example.

```
//ImportsRef is the Imports service namespace
public ImportsRef.ImportFlatFileProfile GetFlatProfile(string sName)
{
    ImportsRef.Imports oImports = new ImportsRef.Imports();
    ImportsRef.ImportProfile oRet = new ImportsRef.ImportProfile();
    ImportsRef.ImportType[] oaImpType = { ImportsRef.ImportType.FlatFile };
    //m_SessionID stores SessionInfo.SessionID
    ImportsRef.ImportProfileInfo[] oaImpProfInfo = oImports.EnumImportProfileInfos
(m_SessionID, oaImpType);
    foreach (ImportsRef.ImportProfileInfo impProfInfo in oaImpProfInfo)
    {
        ImportsRef.ImportProfile oImpProf = oImports.GetImportProfile(m_SessionID,
impProfInfo.ImportID);
```

```
        if (oImpProf.Name == sName)
        {
            oRet = oImpProf;
            break;
        }
    }
    //cast ImportProfile to ImportFlatFileProfile
    return (ImportsRef.ImportFlatFileProfile)oRet;
}
```

Setting a Flat File Profile's Properties

Use the ImportFlatFileProfile data class to set flat file profile properties such as the column delimiter. The following C# function sets an import profile's `ColumnDelimiter`, `StripQuotedStrings`, and `TrimSpaces` properties.

Note:

ImportFlatFileProfile inherits from the ImportProfile class.

```
//ImportsRef is the Imports service namespace
public ImportsRef.ImportFlatFileProfile SetFlatFileProps(Int32 nProfId, char cDelim,
bool bStripQuotes, bool bTrim)
{
    ImportsRef.Imports oImports = new ImportsRef.Imports();
    //m_SessionID stores SessionInfo.SessionID
    ImportsRef.ImportProfile oImpProf = oImports.GetImportProfile(m_SessionID,nProfId);
    //cast ImportProfile to ImportFlatFileProfile
    ImportsRef.ImportFlatFileProfile oImpFlatProf = (ImportsRef.ImportFlatFileProfile)
oImpProf;
    oImpFlatProf.ColumnDelimiter = cDelim;
    oImpFlatProf.StripQuotedStrings = bStripQuotes;
    oImpFlatProf.TrimSpaces = bTrim;
    oImports.SaveImportProfile(m_SessionID,oImpFlatProf);
    return oImpFlatProf;
}
```

Setting Dimensions and Properties in an Import Profile

You can specify dimensions and properties to be imported through the `DimensionMappings` field of an ImportProfile object. This works as follows:

- The `DimensionMappings` field points to an array of ImportDimensionMapping objects.
- Each ImportDimensionMapping object in the array defines the import of one dimension. To specify the dimension into which a data source dimension is imported, use the `SourceDimension` and `TargetDimension` fields.

  Use the `ImportDimensionMapping.ProcessType` field to specify whether the imported dimension is new, or is to merge with or replace an existing dimension.

- The `ImportDimensionMapping.PropertyMappings` field points to an array of ImportPropertyMapping objects. Each array object represents a property of the dimension.
- Each ImportPropertyMapping object in the array defines the import of one property. To specify the property into which a property in the data source is imported, use the `SourceProperty` and `TargetProperty` fields.

The following C# function uses `EnumFlatFileDimensions` to obtain the dimensions in a flat import file and add the dimensions and their properties to an import profile.

```
//ImportsRef is the Imports service namespace
public Int32 MapDimsProps(ImportsRef.ImportProfile oImpProfile, string sFile,
ImportsRef.ImportMappingType oImpMapType)
{
    ImportsRef.Imports oImports = new ImportsRef.Imports();
    ImportsRef.ImportType[] oaImpType = { ImportsRef.ImportType.FlatFile };
    //m_SessionID stores SessionInfo.SessionID
    ImportsRef.Dimension[] oDimArr = oImports.EnumFlatFileDimensions(m_SessionID, sFile,
oImpProfile.ImportID);
    ImportsRef.ImportDimensionMapping[] oImpDimMaps = new
ImportsRef.ImportDimensionMapping[oDimArr.Length];
    for (int i = 0; i < oDimArr.Length; i++)
    {
        //map the dimensions in the import file
        ImportsRef.Dimension oDim = new ImportsRef.Dimension();
       ImportsRef.ImportDimensionMapping oDimMap = new ImportsRef.ImportDimensionMapping
();
        oDim = oDimArr[i];
        oDimMap.SourceDimension = oDim;
        oDimMap.TargetDimension = oDim;
        //map the properties
        ImportsRef.PropertyDefinition[] oaPropDefs =
oImports.EnumFlatFileDimensionProperties(m_SessionID, sFile, oDim.Name,
oImpProfile.ImportID);
        ImportsRef.ImportPropertyMapping[] oaImpPropMaps = new
ImportsRef.ImportPropertyMapping[oaPropDefs.Length];
        for (int j = 0; j < oaPropDefs.Length; j++)
        {
            ImportsRef.PropertyDefinition oPropDef = new ImportsRef.PropertyDefinition
();
            ImportsRef.ImportPropertyMapping oImpPropMap = new
ImportsRef.ImportPropertyMapping();
            oPropDef = oaPropDefs[j];
            oImpPropMap.SourceProperty = oPropDef;
            oImpPropMap.TargetProperty = oPropDef;
            oaImpPropMaps[j] = oImpPropMap;
        }
        //add the array of propDefs to the dimensionMapping object for the dimension
        oDimMap.PropertyMappings = oaImpPropMaps;
        //set the specified type of import
        oDimMap.ProcessType = oImpMapType;
        //add the dimensionMapping object to the array of dimensionMapping objects
        oImpDimMaps[i] = oDimMap;
    }
    oImpProfile.DimensionMappings = oImpDimMaps;
    oImports.SaveImportProfile(m_SessionID, oImpProfile);
    Int32 nDimMapLength = oImpProfile.DimensionMappings.Length;
```

```
    return nDimMapLength;
}
```

Importing from a Flat File

To import from a flat file, you must upload the file to the server, then use `StartFlatFileImportJob` to run an import job. If you want to monitor the job status, use the Jobs service class.

Note:

To upload the file, you can use `UploadImportFlatFile` or perform a buffered upload with the Utilities class. See "Performing a Buffered Upload" on page 41.

To access the results of the import, use the ImportResultSet object returned by `GetImportResults`. Before calling `GetImportResults`, you must obtain the `AttachmentID` and `BatchSequenceID` using data classes of the Jobs service class. You then set these IDs in the corresponding fields of the JobAttachmentBatchLocator object passed to `GetImportResults`.

The following C# function runs an import job and returns the ImportResultSet object that contains the import results.

Note:

To obtain the `AttachmentID` and `BatchSequenceID` IDs needed for the JobAttachmentBatchLocator object, the example calls the user-defined function listed in "Processing an Import Job" on page 40.

```
public ImportsRef.ImportResultSet ImportFlatFileUtilitiesTransfer(Int32 nAppId, string
sFile, ImportsRef.ImportProfile oImpProf)
        {
            //ImportsRef is the namespace for the Imports web service
            ImportsRef.Imports oImports = new ImportsRef.Imports();
            ImportsRef.ApplicationLocator oAppLoc = new ImportsRef.ApplicationLocator();
            oAppLoc.ApplicationID = nAppId;
            //m_SessionID stores SessionInfo.SessionID
            oAppLoc.SessionID = m_SessionID;
            //m_LibraryId stores the ID of the master library
            oAppLoc.LibraryID = m_LibraryId;
            //string sUploadedFile = oImports.UploadImportFlatFile(m_SessionID, sFile);
            string sUploadedFile = this.UploadFile(sFile);
            ImportsRef.ServerJob oJob = oImports.StartFlatFileImportJob(oAppLoc,
oImpProf.ImportID, sUploadedFile, null);
            ImportsRef.JobAttachmentBatchLocator oBatchLoc = new
ImportsRef.JobAttachmentBatchLocator();
            oBatchLoc.SessionID = m_SessionID;
            oBatchLoc.JobID = oJob.JobID;
            //process job and return attachment ids from user-defined function
            Int32[] nAttachIds = this.ProcessImportJob(oJob.JobID);
```

```
            oBatchLoc.AttachmentID = nAttachIds[0];
            oBatchLoc.BatchSequenceID = nAttachIds[1];
            ImportsRef.ImportResultSet oResult = oImports.GetImportResults(oBatchLoc);
            return oResult;
        }
```

Processing an Import Job

You can use the Jobs class to process the job initiated by `Imports.StartImportJob` and to get the job attachment and batch sequence IDs for the JobAttachmentBatchLocator object passed to `Imports.GetImportResults`.

The following C# function processes a job and returns its attachment and batch sequence IDs.

```
public Int32[] ProcessImportJob(Int32 nJobID)
{
    //JobsRef is the namespace for the Jobs service
    JobsRef.Jobs oJobs = new JobsRef.Jobs();
    JobsRef.JobStatus oJobStatus = new JobsRef.JobStatus();
    do
    {
        Thread.Sleep(2000);
        //m_SessionID stores SessionInfo.SessionID
        oJobStatus = oJobs.GetJobStatus(m_SessionID, nJobID);
    }while ((oJobStatus != JobsRef.JobStatus.Completed) && (oJobStatus !=
JobsRef.JobStatus.Aborted));
    //filter for the job
    JobsRef.JobFilter oJobFilter = new JobsRef.JobFilter();
    oJobFilter.JobID = nJobID;
    oJobFilter.FromSubmittedTime = DateTime.Today;
    oJobFilter.ToSubmittedTime = DateTime.Today;
    //get the JobInformation object for the job
    JobsRef.JobInformation[] oJobInfos = oJobs.EnumJobs(m_SessionID, oJobFilter).Jobs;
    JobsRef.JobInformation oJobInfo = oJobInfos[0];
    //Return the job attachment and batch sequence IDs
    Int32 nAttachId = oJobInfo.JobAttachments[0].AttachmentID;
    Int32 nSeqId = oJobInfo.JobAttachments[0].BatchSequenceIDs[0];
    Int32[] naRet = { nAttachId, nSeqId };
    return naRet;
}
```

12 Utilities Web Service

In This Chapter

The Utilities class provides helper methods related to importing with the Imports class and to extracting transactions. For example, the Utilities class provides methods to perform a buffered upload of an import file.

The following topics show how to perform some tasks related to the Utilities class.

Performing a Buffered Upload

The following C# function shows how to perform a buffered file upload with `StartFileUpload`, `Transfer`, and `EndTransfer`.

```
private string UploadFile(string sFilePath)
{
    int idx = sFilePath.LastIndexOf("\\");
    string sFileName = sFilePath.Substring(idx + 1, (sFilePath.Length - (idx + 1)));
    //UtilitiesRef is the Utilities service namespace
    UtilitiesRef.Utilities oUtils = new UtilitiesRef.Utilities();
    //m_SessionID stores SessionInfo.SessionID
    string sTransferFile = oUtils.StartFileUpload(m_SessionID, sFileName);
    FileStream fs = new FileStream(sFilePath, FileMode.Open, FileAccess.Read);
    byte[] buf = new byte[fs.Length];
    fs.Read(buf, 0, (int)fs.Length);
    oUtils.Transfer(m_SessionID, sTransferFile, 0, (int)fs.Length, buf);
    string sUploadedFile = oUtils.EndTransfer(m_SessionID, sTransferFile, false);
    return sUploadedFile;
}
```

Enumerating Valid File Delimiters

`EnumDelimiters` returns an array of objects that represent the valid delimiters. The following C# functions indicates whether a given delimiter is valid.

```
public bool IsValidDelim(char chDelim)
{
    //UtilitiesRef is the Utilities service namespace
```

```
    UtilitiesRef.Utilities oUtilities = new UtilitiesRef.Utilities();
    bool bRet = false;
    //m_SessionID stores SessionInfo.SessionID
    UtilitiesRef.DelimiterInfo[] oDelimInfos = oUtilities.EnumDelimiters(m_SessionID);
    foreach (UtilitiesRef.DelimiterInfo delimInfo in oDelimInfos)
    {
        if (chDelim == delimInfo.Delimiter)
        {
            bRet = true;
            break;
        }
    }
    return bRet;
}
```

13 Compares Web Service

In This Chapter

The Compares service class exposes functionality for comparing Application Views, dimensions, and members. You can compare using the current Application View or a snapshot from a given point in time, and compare against the Master Application or an earlier snapshot.

The following table summarizes commonly-used data classes related to the Compares class.

Data Class	Description
ServerJob	Represents a comparison job, and is returned by methods that start comparisons. You can monitor the progress of the job with the Jobs class; see Chapter 10, "Jobs Web Service."
CompareCriteria	Defines the basis of a comparison.
CompareResultSet	Represents the results of a comparison. You can use CompareResultSet fields to drill down to various comparison-related data classes, such as the CompareDimension class.
CompareDimension	Represents a dimension in a CompareResultSet object. You can access an array of a CompareResultSet object's CompareDimension objects with the `Dimensions` field.
CompareMember	Represents the comparison results for a member.
CompareStatistics	Stores various statistics related to comparison differences, such as counts of removed members and the total number of differences. CompareStatistics is accessed with the `Statistics` field of data classes such as CompareResultSet, CompareDimension, and CompareMember.

The following topics show how to perform some tasks related to the Compares class.

Comparing Application Views

To compare Application Views, use `StartApplicationCompareJob`. When the ServerJob returned by this method finishes processing, obtain the comparison results by passing the ServerJob to `GetCompareResults` and then use the returned CompareResultSet object to drill down to the desired comparison data.

The following C# function launches a job to compare Application Views and returns a CompareResultSet object.

Note:

The function calls the user-defined function `ProcessJob` to processes the job. See "Processing a Job" on page 34.

```
//ComparesRef is the namespace for the Compares web service
public ComparesRef.CompareResultSet GetAppDiffs(Int32 nFromAppId, Int32 nToAppId)
{
    ComparesRef.ApplicationLocator oFromAppLoc = new ComparesRef.ApplicationLocator();
    ComparesRef.ApplicationLocator oToAppLoc = new ComparesRef.ApplicationLocator();
    //m_SessionID stores SessionInfo.SessionID
    oFromAppLoc.SessionID = m_SessionID;
    oFromAppLoc.LibraryID = m_LibraryId;
    oFromAppLoc.ApplicationID = nFromAppId;
    oToAppLoc.SessionID = m_SessionID;
    oToAppLoc.LibraryID = m_LibraryId;
    oToAppLoc.ApplicationID = nToAppId;
    ComparesRef.Compares oCompares = new  ComparesRef.Compares();
    ComparesRef.CompareCriteria oCompareCriteria = new ComparesRef.CompareCriteria();
    oCompareCriteria.CompareProperties = true;
    ComparesRef.ServerJob oJob = oCompares.StartApplicationCompareJob
(oFromAppLoc,oToAppLoc,oCompareCriteria);
    //custom function to process the job
    bool bRet = this.ProcessJob(oJob.JobID);
    ComparesRef.CompareResultSet oResults = oCompares.GetCompareResults(oJob);
    return oResults;
}
```

Evaluating Comparison Results

A CompareResultSet object represents the results of a comparison job. You can obtain data from CompareResultSet object fields such as `Statistics`, or find dimensions that differ using the `Dimensions` field. You can drill down to members with the `EnumChildCompareMembers` method.

The following topics provide examples of how to perform these result evaluation tasks.

Finding Dimensions with Changes

The following C# function loops through a CompareResultSet object's CompareDimension objects and returns an array containing the names of all members that differ, including parent member names when applicable. If the `CompareDimension.Statistics.Removes` field is greater than zero, the dimension contains removed members, and the names of these members are obtained with the user-defined function described in "Finding Members with Changes" on page 45.

```
//ComparesRef is the namespace for the Compares web service
public string[] GetRemovedMembers(ComparesRef.CompareResultSet oResults)
{
    string[] sRet = new string[oResults.Statistics.Removes];
    //counter to track items added to return value
```

```
    Int32 nCounter = 0;
    ComparesRef.Compares oCompares = new ComparesRef.Compares();
    ComparesRef.CompareDimension[] oCompDims = oResults.Dimensions;
    for (Int32 i = 0; i < oCompDims.Length;i++)
    {
        //if the dimension has changes....
        if (oCompDims[i].Statistics.Removes > 0)
        {
            //get changed members with custom function
            string[] sTest = this.GetRemovedChildren(oResults, oCompDims
[i].Root.CompareMemberID, string.Empty);
            for (Int32 j = 0; j < sTest.Length; j++)
            {

                sRet[nCounter] = sTest[j];
                nCounter = nCounter + 1;
            }
        }
    }
    return sRet;
}
```

Finding Members with Changes

The following C# function takes a CompareResultSet object, the `CompareMemberID` of the CompareMember object containing the members to be processed, and the parent member's name, and returns the names of all removed child members of the parent. `EnumChildCompareMembers` returns an array of CompareMember objects representing any child members that differ. The function recurses if a CompareMember object contains removed children.

```
//ComparesRef is the namespace for the Compares web service
public string[] GetRemovedChildren(ComparesRef.CompareResultSet oResults, Int32
nCompMemId, string sPar)
{
    ComparesRef.Compares oCompares = new ComparesRef.Compares();
    ComparesRef.CompareMemberLocator oCompareMemLoc = new
ComparesRef.CompareMemberLocator();
    Int32 nCounter = 0;
    oCompareMemLoc.CompareMemberID = nCompMemId;
    oCompareMemLoc.CompareResultSetID = oResults.CompareResultID;
    //m_SessionID stores SessionInfo.SessionID
    oCompareMemLoc.SessionID = m_SessionID;
    ComparesRef.CompareMember[] oCompareMems = oCompares.EnumChildCompareMembers
(oCompareMemLoc);
    Int32 nLength = 0;
    foreach (ComparesRef.CompareMember compareMem in oCompareMems)
    {
        nLength = nLength + compareMem.Statistics.Removes;
    }
    string[] sRet = new string[nLength];
    for (Int32 i = 0; i < oCompareMems.Length; i++)
    {
        //if member differs, add to return value
        if (oCompareMems[i].DifferenceType == ComparesRef.CompareDifferenceType.Remove)
```

```
        {
            sRet[nCounter] = sPar + oCompareMems[i].SourceMember.Name;
            nCounter = nCounter + 1;
        }
        //recurse to get child member changes
        if (oCompareMems[i].HasChildren == true && oCompareMems[i].Statistics.Removes>
0)
        {
            string[] sTemp = this.GetRemovedChildren(oResults, oCompareMems
[i].CompareMemberID, sPar + oCompareMems[i].SourceMember.Name + ".");
            for (int j = 0; j < sTemp.Length; j++)
            {
                sRet[nCounter] = sTemp[j];
                nCounter = nCounter + 1;
            }
        }
    }
    return sRet;
}
```

Index

E

F

G

H

I

T

U

W