SDK Documentation

Version 2.0

Contents

1 Document control

1.1 About this document

Author	S.S.
Status	Version 2.0

1.2 Changes made to this document

Version	Sections changed or created	Author or contributor name	Date	Change summary
1.0	All	S.S.	11 Oct 2007	Document created
2.0	Update for new APIs	S.S.	18 Apr 2010	

2 Purpose of this document

This document is to serve the purpose of a SDK documentation for utilizaing the Analyzer 2005 web services APIs provided by Strategy Companion.

1. Analyzer 2005 Classes and Enumerations available for programming with Analyzer APIs
2. Brief Description and usage of the Web Service APIs available along with sample code.

3 Analyzer 2005 classes

3.1 Analyzer Enums

This section explains the various Enumerations available for programming with Analyzer APIs.

3.1.1 MeasureTypes

This Enum represent the various types of measures.

Name
Regular
KPIAll
KPIValue
KPIGoal
KPIStatus
KPITrend

3.1.2 DimensionTypes

This Enum represent the various types of dimensions.

Name
Hierarchy
Dimension

3.1.3 RoleMemberTypes

This Enum represent the various types of Role Members.

Name
User
Role

3.1.4 SecurityMemberTypes

This Enum represents various types of Security Members.

Name
User
Group
Role

3.1.5 CatalogItemTypes

This Enum represent the various types of catalog items.

Name
Folder
Report
DataSource

3.1.6 SystemLanguages

This Enum represent the languages that are supported by Analyzer.

Name
English
TradChinese
SimpChinese

3.1.7 RowsPerPage

This Enum represent the options available for setting the display of rows per page property.

Name
Fifteen
Thirty
Fifty

3.1.8 DefaultHomePages

This Enum represent the options for configuring default home page in Analyzer application.

Name
WorkingArea
SharedReports
PersonalReports
KPIWatchList

3.1.9 DataSourceType

This Enum represent the data source types specifying the version of SQL Server Analysis Services.

Name
SqlAs2000
SqlAs2005

3.1.10 ToolbarButton

This Enum represent the list of toolbar buttons that can be configured in Analyzer application.

Name
DataSource
SchemaTree
ObjectPanel
BookmarkPanel
KPIPanel
LanguagePanel
Save
SaveAs
SendMsg
CreateBookmark
UpdateBookmark
Close
Config
PlayVideo
Refresh

ZoomIn
ZoomOut
ResetZoom
SwitchBorder
Excel
Print
EntireToolbar

3.1.11 PermissionState

This Enum represent the different access levels that can be set for each permission type.

Name
NotSpecified
Grant
Deny

3.1.12 SubscriptionType

This Enum represent the different subscription types.

Name
Time
DataDriven

3.2 Overview of to Analyzer Classes

Following is the class diagram describing the classes available in Analyzer Web Services.

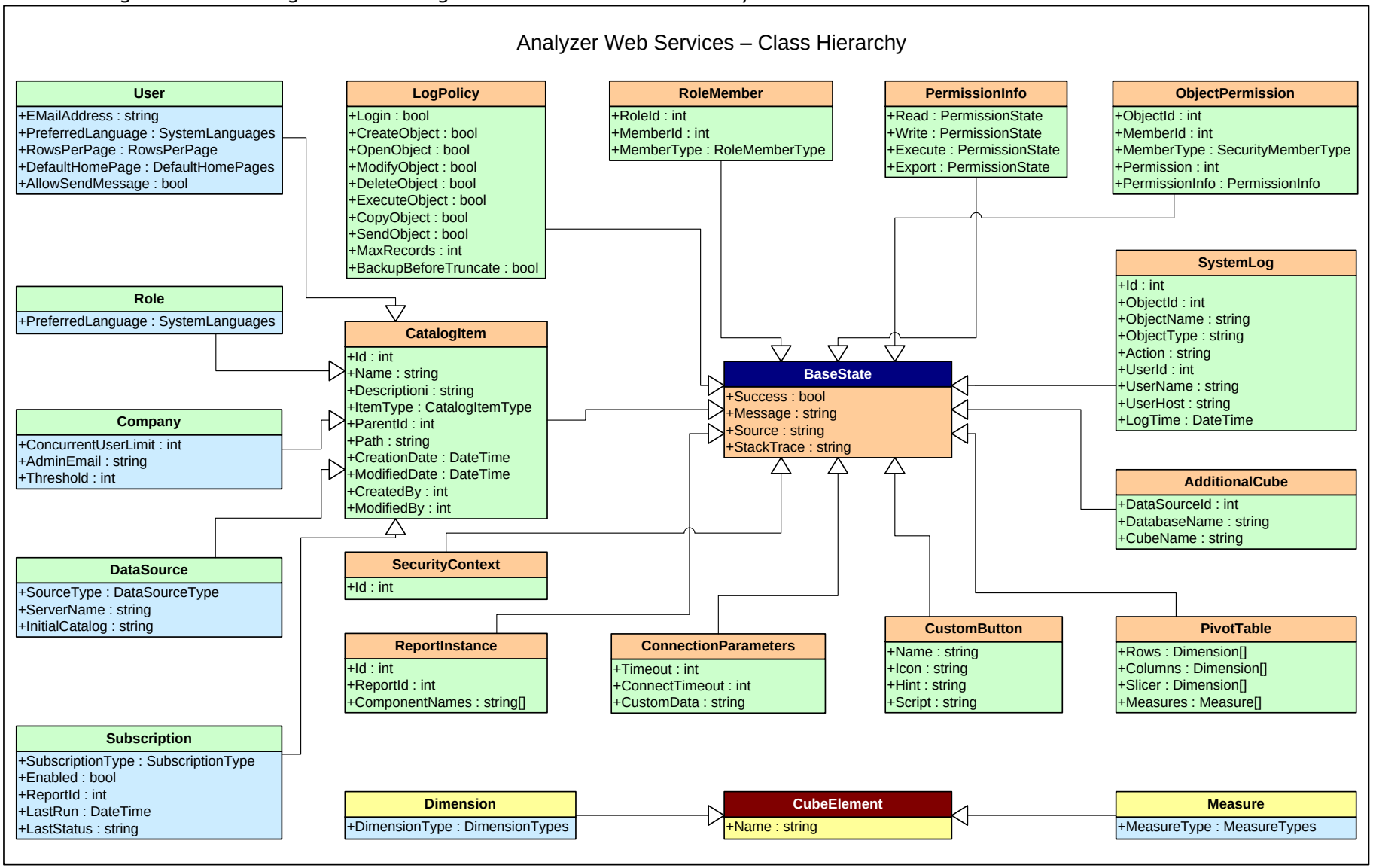

Figure 1: Analyzer Class Hierarchy

The above class diagram provides the list of all the classes which can be used while consuming the Analyzer 2005 web services. The color code in the diagram is provided for identifying the base class and the inherited class. The objects of a class and the header of the derived classes have the same color. For example, all the objects of the class BaseState have the same background color as of the headers for all the classes that are inherited from the class BaseState (Ex: ReportInstance, RoleMember etc).

The following section explains each of the classes available.

3.2.1 BaseState

This is the base class for most of the classes available in the web service APIs. All the other classes derived from this class will have all the properties of this class.

Properties of the class BaseState are mentioned below.

CatalogItem:BaseState		
Name	**Type**	**Description**
Success	bool	This property will contain the flag value, whether there is any exception that had occurred. If there is an exception, then this value will be set to false.
Message	String	This property will contain any description set for the class. If an exception occurs, then this property will have the description on the particular exception.
Source	String	This property will contain the source object name where the exception occurred.
StackTrace	String	This property will contain the entire stack trace where the exception occurred.

3.2.2 CatalogItem

The class CatalogItem is used for representing different Catalog Items of the Analyzer 2005 objects. Some of the examples are Folder, Report, User, Role etc. This class is the base class for the main catalog components like User, Role and DataSource.

CatalogItem:BaseState		
Name	**Type**	**Description**
Id	int	Contain the Id of the class object like UserId, RoleId or DataSource Id, Folder Id etc...
Name	String	Contains the name of the object
Description	String	Contains the description of the object which is provided during creation of the specific CatalogItem.
ItemType	CatalogItemType	This field indicates the type of the CatalogItem whether User or Role or DataSource.
ParentId	Int	Contains the Id of the parent element. For example, if the CatalogItem is a folder created by user, then ParentId will contain the parent folder Id of this folder.
Path	String	
CreationDate	DateTime	Contains the date when the CatalogItem was created.
ModifiedDate	DateTime	Contains the date when the CatalogItem was last modified.
CreatedBy	Int	Contains the UserId who created the CatalogItem
ModifiedBy	Int	Contains the UserId who last modified the CatalogItem.

3.2.3 User

The class User is used to represent the users logged into the Analyzer application. It will have the details about customization details of User along with the details mentioned in the class CatalogItem.

User:CatalogItem		
Name	**Type**	**Description**
EMailAddress	String	Contains Email addres of the user.
PreferredLanguage	SystemLanguages	The language in which all the contents to be displayed for this user.
RowsPerPage	RowsPerPage	Contains the number of rows the user can see while listing reports, users etc.
DefaultHomePage	DefaultHomePages	Contains the default home page that the user will see when logs in.
AllowSendMessage	Bool	Indicates whether the user is allowed to send message or not.

3.2.4 Company

The class Company is used to represent the Company object which is associated with Analyzer report. This class will contain the Company level details like the company E-mail contact etc.,.

Company:CatalogItem		
Name	**Type**	**Description**
ConcurrentUserLimit	Int	This is the number of simultaneous allowed logins for the Company.Company. Ex: If there are 50 users for a company and the ConcurrentUserLimit is set to 30, then at any given point of time, only 30 users from that company can login into Analyzer application.
AdminEmail	String	This field will contain the e-mail address for the company Admin for the Analyzer Reports.
Threshold	Int	This the number of simultaneous logins from users of the same company, when reached the limit, an e-mail alert will be sent to the Company Admin indicating that the threshold login has been reached. Ex: If the Threshold set for a company is 20 and number of simultaneous login reaches this

		nunmber, an e-mail will be sent to the company Admin.

3.2.5 Role

The class Role is used to represent the the Roles in Analyzer application. It will have the details about customization details of Role along with the details mentioned in the class CatalogItem.

Role:CatalogItem		
Name	**Type**	**Description**
RoleId	Int	The RoleId of the role object which is being represented.
MemberId	Int	The MemberId associated with the Role.
MemberType	SecurityMemberType	To represent the Member Type whether User or Role.

3.2.6 DataSource

The class DataSource is used to represent the DataSources in the Analyzer application. It will have the details about customization details of User along with the details mentioned in the class CatalogItem.

DataSource:CatalogItem		
Name	**Type**	**Description**
SourceType	DataSourceType	Indicates the Analysis Server version whether 2000 or 2005.
ServerName	String	Contains the name of the Analysis Server.
InitialCatalog	String	Contains the default databse accessibe for this DataSource.

3.2.7 RoleMember

The class RoleMember is used to represent a specific combination of Role and Member in Analyzer application.

RoleMember:BaseState		
Name	**Type**	**Description**
RoleId	Int	The RoleId of the role object which is being represented.
MemberId	Int	The MemberId associated with the Role.
MemberType	SecurityMemberType	To represent the Member Type whether User or Role.

3.2.8 SecurityContext

The class SecurityContext is used to represent the User Context under which all the actions are performned using Analyzer APIs.

SecurityContext:BaseState		
Name	**Type**	**Description**
Id	Int	Id of the SecurityContext object.

3.2.9 SystemLog

The class SystemLog is used to represent a single system log entry in Analyzer application. This class will have the necessary properties required for generating System Log report.

SystemLog:BaseState		
Name	**Type**	**Description**
Id	Int	The Unique Id for each log entry in the System Log
ObjectId	Int	The Id of the object on which the action performed.
ObjectName	String	The Name of the object on which the action performed.
ObjectType	String	The Object Type of the object on which the action performed.
Action	String	The action performed on the object.
UserId	Int	The Id of the user who performed the action.
UserName	String	Name of the user who performed the action.
UserHost	String	The server where the action was performed.
LogTime	Datetime	The time when the action was performed.

3.2.10 LogPolicy

The class LogPolicy is used to represent the Log Policy in Analyzer application. This class explains on how the System Log policy is set.

LogPolicy:BaseState		
Name	**Type**	**Description**
Login	bool	Flag indicating whether to create a log entry during Login
CreateObject	bool	Flag indicating whether to create a log entry when an object is created
OpenObject	bool	Flag indicating whether to create a log entry when an object is accessed
ModifyObject	bool	Flag indicating whether to create a log entry when an object is modified
DeleteObject	bool	Flag indicating whether to create a log entry when an object is deleted
ExecuteObject	bool	Flag indicating whether to create a log entry when an object is executed
CopyObject	bool	Flag indicating whether to create a log entry when an Saved As action is performed
SendObject	bool	Flag indicating whether to create a log entry when a message is sent
MaxRecords	Int	Indicates the maximum number of log entries that can exist in the Log.
BackBeforeTruncate	Bool	Flag indicating whether to backup the log details before deleting the log once the number of entries in the log exceeds maximum size.

3.2.11 ObjectPermission

The class ObjectPermission is used to represent the permission on a Catalog Item for a specific Member in Analyzer application. This class will have the object on which the permission is set and the member for whom the permission is set along with the permission details.

ObjectPermission:BaseState		
Name	**Type**	**Description**
ObjectId	Int	Id of the Object on which the permission is being set.
MemberId	Int	Id of the member for whom the permission is being set.
MemberType	SecurityMemberType	Member Type representing if the member is of type User or Role.
Permission	Int	A specific number representing granted permissions. 85 being the highest indicating that the member has permission to perform all the operations List, Execute, Export and Config. Following are the details on different permissions. 1 – List only 4 – Config only 5 – List and Config 16 – Execute only 17 – List and Execute 20 – Execute and Config 64 – Export only 65 – List and Export 68 – Export and config 80 – Execute and Export 81 – List, Execute and Export 84 – Execute, Export and Config 85 – List, Execute, Export and Config
PermissionInfo	PermissionInfo	PermissionInfo object which the corresponding permission details.

3.2.12 PermissionInfo

The class PermissionInfo is used to represent set the permissions in Analyzer application. Each of the permission type (Read, Write, Execute, Export) will have individual access status (NotSpecified, Grant, Deny).

PermissionInfo:BaseState		
Name	**Type**	**Description**
Read	PermissionState	Indicates the Read permission status.
Write	PermissionState	Indicates the Read permission status.
Execute	PermissionState	Indicates the Read permission status.
Export	PermissionState	Indicates the Read permission status.

3.2.13 CustomButton

The class CustomButton is used to represent and object of the custom button that can be configured into the Analyzer UI. For example, if you want to dispay your own custom button in the Analyzer screen, you can do it by creating an object of Custom Button.

CustomButton:BaseState		
Name	**Type**	**Description**
Name	String	Name of the custom button
Icon	String	Relative path of the image for custom button
Hint	String	Tooltip for the custom button
Script	String	The javascript action that needs to be executed on click of the custom button

3.2.14 ReportInstance

The class ReportInstance is used to represent an instance of Analyzer Report.

ReportInstance:BaseState		
Name	**Type**	**Description**
Id	String	Id of the ReportInstance object.
ReportId	Int	Id of the report which is represented by the Report Instance.
Components	String	The components that are part of the report.

3.2.15 AdditionalCube

The class AdditionalCube is used to represent an instance of cube that can be added to the Analyzer Report.

AdditionalCube:BaseState		
Name	**Type**	**Description**
DataSourceId	Int	Id of the DataSource where the cube database is present.
DatabaseName	String	Database Name where the cube is present
CubeName	String	Name of the cube which needs to be added to the report.

3.2.16 CubeElement

The class CubeElement is used to represent the various elements of the cube.

CubeElement:BaseState		
Name	**Type**	**Description**
Name	String	Name of the cube element

3.2.17 Dimension

The class Dimensioin is used to represent a dimension in a cube.

Dimension:CubeElement		
Name	**Type**	**Description**
DimensionType	DimensionTypes	Indicates the type of dimension, whether hierarchy or attribute dimension

3.2.18 Measure

The class Measure is used to represent a measure in a cube.

Dimension:CubeElement		
Name	**Type**	**Description**
MeasureType	MeasureTypes	Indicates the type of measure, whether regular or KPI related.

3.2.19 Subscription

The class Subscription is used to represent an details of a subscription associated with a report.

Subscription:BaseState		
Name	**Type**	**Description**
SubscriptionType	SubscriptionType	Subscription type to indicate if the subscription is data driven or schedule specific based on time
Enabled	Bool	Boolean value indicating whether the subscription is enabled or not
ReportId	Int	ReportId of the report on which the subscription is defined.
LastRun	DateTime	The data when the subscription was triggered last time
LastStatus	String	The result of the last execution of the subscription.

3.2.20 ConnectionParameters

The class ConnectionParameters is used to represent connection timeout related attributes that can be added to the Analyzer Report.

AdditionalCube:BaseState		
Name	**Type**	**Description**
Timeout	Int	Number of seconds for command timeout. This property also determines the maximum time that the instance should wait for an update to a writeback table to be successful before returning an error. This property can help you to avoid very long runs until the queries are optimized to run faster.
ConnectTimeout	Int	Determines the maximum amount of time the client application will attempt to connect to the server before timing out
CustomData	String	Specifies information that can be retrieved in SSAS scripts by using MDX function CustomData().

4 Analyzer APIs

The Adventure Works DW OLAP database provided by Microsoft during SQL Server Analysis Services installation has been used in the sample code that is provided for the APIs.

4.1 Initialize Analyzer Web Services object

4.1.1 Create Proxy class for the asmx file

The typical way a web services used in dot net application is by adding a web reference to a project which is suppose to consume the web services. But this will have a code maintenance task if you keep changing your web server that is hosting the web methods. Hence it is always a good practice to create a proxy class (C# or VB.Net) when you want to consume a web service. This way you can specify the web server name before consuming any of the web methods by assigning the value of url with appropriate web server which is explained in further section.

You can use the utility WSDL.exe for creating a proxy class and then add this class to your project under the same namespace of your project where you want to consume the web service.

Following is the command line to create a proxy for Analyzer2005 classes in C#. <AnalyzerServer> is the fully qualified name of the server where Analyzer application is installed. The output proxy file named Analyzer2005.cs will be created C:\ drive when the following command is executed.

```
wsdl.exe /l:CS /out:c:\Analyzer2005.cs http://<AnalyzerServer>/Services/Analyzer2005.asmx?WSDL
```

4.1.2 Initializing Analyzer

Before you start using any of the Analyzer APIs, you must initialize the web service object. You can have Analyzer setup in two modes of IIS authentication. Its Basic Authentication and Integrated Windows Authentication. You can use Basic Authentication if the end user is on the internet so that the credentials can be passed to the Analyzer Web Service call which is required to access Microsoft Analysis Services. If the end user is within the same domain as of Anlayzer Web Server, then you can use Integrated Windows Authentication as the windows login credentials will be used to access Analyzer Web Services.

Following code explains how to initialize Analyzer object based on the Authentication. This section also explains the code if the web services are used to display Analyzer user interface in web browser (IE 5.0 onwards).

```
private Analyzer2005 InitializeAnalyzer()
      {
         bool isBasic = true; // set the flag based on the authentication set in IIS for analyzer
                              // if the callilng application is a web applicaton,
                              // then even that needs to be set to the same authentication
                              // you can have these values in the configuration file
         Analyzer2005 analyzerObject = new Analyzer2005();

         // use the following if contition check if the caller is a web application

         if (Session.IsCookieless)
         {
             string analyzerPath = string.Format(
                  "{0}/{1}/({2})",
                  "Analyzer_Web_Server", // the machine where analyzer is installed. Ex: MACHINE1.DOMAIN
                 "AnalyzerVirtualDirectoryPath", // the virtual directory name of Analyzer application.
                                                 // For example : Analyzer
                 Session.SessionID);
         }
         else
         {
             string analyzerPath = string.Format(
                 "{0}/{1}",
                 "Analyzer_Web_Server", // the machine where analyzer is installed. Ex: MACHINE1.DOMAIN
                 "AnalyzerVirtualDirectoryPath"); // the virtual directory name of Analyzer application.
                                                  // For example : Analyzer
         }
```

```
analyzerObject.Url = string.Format("http://{0}/services/Analyzer2005.asmx", analyzerPath);

if (isBasic)
{
    // For basic authentication
    string userid = "user1"; // if the calling application is web application,
                             // then use Request.ServerVariables["AUTH_USER"]
    string password = "password1"; // if the calling application is web application,
                            // then use Request.ServerVariables["AUTH_PASSWORD"]
    CredentialCache cache = new CredentialCache();
    cache.Add(
        new Uri(analyzerObject.Url),
        "Basic",
        new NetworkCredential(userid, password)
        );
    analyzerObject.PreAuthenticate = true;
    analyzerObject.Credentials = cache;
}
else
{
    // For integrated windows authentication
    analyzerObject.Credentials = CredentialCache.DefaultCredentials;
}

// use the following piece of code if the caller is a web application
analyzerObject.CookieContainer = new CookieContainer();
foreach (string key in Request.Cookies.AllKeys)
```

```
            {
                string domain = "MACHINE.DOMAIN"; // you can have these values in the configuration file
                analyzerObject.CookieContainer.Add(new Cookie(Request.Cookies[key].Name,
                     Request.Cookies[key].Value, "/", domain));
            }
            return analyzerObject;
        }
```

4.1.3 Initializing Analyzer

The following code shows how to invoke the Initialize() method that is described above.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
```

4.2 User Session Management APIs

4.2.1 LogonUser Method

This method sets up the security context for a specific user on Analyzer Session. The securitycontext object returned by this method call can be used in many of the Analyzer Web Services method like CreateDataSource, AddObjectPermission etc…

Syntax:

```
LogonUser(string domain_name, string user_name, string password, int expiration)
```

Arguments:

```
domain_name
```

The IIS domain name where the Analyzer application is setup. To find out this value, go to IIS → Web Sites → Analyzer Virtual Directory → Properties → Directory Securtiy → Default Domain

```
user_name
```

The network user name through which the web service requests will be made.

```
Password
```

Network password for the user_name passed.

```
expiration
```

To specify the session duration the user session will be active for. -1 means indefinite duration.

Return Type

This method returns an object of SecurityContext which can be passed into the other Analyzer web service calls.

Examples:

The following statement will return the SecurityContext object valid for indefinite time.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SecurityContext secContext = analyzerObject.LogonUser("PRODDOMAIN", "kmiller", "abcde123", -1)
```

The following statement will return the SecurityContext object valid for 2 minutes.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SecurityContext secContext = analyzerObject.LogonUser("PRODDOMAIN", "kmiller", "abcde123",      120000)
```

4.2.2 LogonUserWithCompanyId Method

This method sets up the security context for a specific user on Analyzer Session for accessing resources specific to a Company ID registered in Analyzer application. The securitycontext object returned by this method call can be used in many of the Analyzer Web Services method like CreateDataSource, AddObjectPermission etc…

Syntax:

```
LogonUser(string domain_name, string user_name, string password, int    companyId, int expiration)
```

Arguments:

```
domain_name
```

The IIS domain name where the Analyzer application is setup. To find out this value, go to IIS → Web Sites → Analyzer Virtual Directory → Properties → Directory Securtiy → Default Domain

```
user_name
```

The network user name through which the web service requests will be made.

```
Password
```

Network password for the user_name passed.

```
companyId
```

Company ID to restrict the resource access specific to a company that is registered in Analyzer Application.

```
expiration
```

To specify the session duration the user session will be active for. -1 means indefinite duration.

Return Type

This method returns an object of SecurityContext which can be passed into the other Analyzer web service calls.

Examples:

The following statement will return the SecurityContext object valid for indefinite time.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SecurityContext secContext = analyzerObject.LogonUser("PRODDOMAIN", "kmiller", "abcde123",  100001, -1)
```

The following statement will return the SecurityContext object valid for 2 minutes.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SecurityContext secContext = analyzerObject.LogonUser("PRODDOMAIN", "kmiller", "abcde123", 100001, 120000)
```

4.2.3 LogonUserWithRoles Method

This method sets up the security context for a specific user on Analyzer Session with access privileges of specific Role. The securitycontext object returned by this method call can be used in many of the Analyzer Web Services method like CreateDataSource, AddObjectPermission etc…

Syntax:

```
LogonUser(string domain_name, string user_name, string password, string       roles, int expiration)
```

Arguments:

```
domain_name
```

The IIS domain name where the Analyzer application is setup. To find out this value, go to IIS → Web Sites → Analyzer Virtual Directory → Properties → Directory Securtiy → Default Domain

```
user_name
```

The network user name through which the web service requests will be made.

```
Password
```

Network password for the user_name passed.

```
Roles
```

Role name to provide access privileges of specific role.

```
Expiration
```

To specify the session duration the user session will be active for. -1 means indefinite duration.

Return Type:

This method returns an object of SecurityContext which can be passed into the other Analyzer web service calls.

Examples:

The following statement will return the SecurityContext object valid for indefinite time.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SecurityContext secContext = analyzerObject.LogonUser("PRODDOMAIN", "kmiller", "abcde123",
'Administrator', -1)
```

The following statement will return the SecurityContext object valid for 2 minutes.

```
SecurityContext secContext = LogonUser("PRODDOMAIN", "kmiller", "abcde123", 100001, 'Report   Designer', 120000)
```

4.2.4 LogoffUser Method

This method closes the session for a given security context.

Syntax:

```
LogoffUser(SecurityContext security_context)
```

Arguments:

```
security_context
```

The security context object which has active session on.

Return Type

This method returns an object of SecurityContext which can be passed into the other Analyzer web service calls.

Examples:

The following statement will return the SecurityContext object valid for indefinite time.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SecurityContext secContext = analyzerObject.LogonUser("PRODDOMAIN", "kmiller", "abcde123", 120000)
      //// Do something
      //// Do something
secContext = LogoffUser(secContext)
```

4.3 Role Management APIs

4.3.1 CreateRole Method

This method creates a new role based on the with the Role object passed in as parameter.

Syntax:

```
CreateRole(SecurityContext context, Role role)
```

Arguments:

```
context
```

The security context under which the CreateRole action needs to be performed.

```
role
```

The role object having the details of new role to be created

Return Type

This method returns an object of Role that got created by CreateRole method.

Examples:

The following code creates a role named "TestRole" and assigns it to the role object "outRole".

```
Role role = new Role();
role.Name = "TestRole";
role.Description = "Testing CreateRole method";
role.PreferredLanguage = SystemLanguages.English;
Analyzer2005 analyzerObject = InitializeAnalyzer();
Role outRole = analyzerObject.CreateRole(secContext, role); // you can also pass null for securitycontext
```

4.3.2 CreateRole Method (Company specific)

This overloaded method creates a new role based on the the Role object passed in as parameter. The new role created will be associated with the Company based on the Company ID passed in as parameter.

Syntax:

```
CreateRole(SecurityContext context, Role role , int companyId)
```

Arguments:

```
context
```

The security context under which the CreateRole action needs to be performed.

```
role
```

The role object having the details of new role to be created

```
companyId
```

The CompanyID to which the new role should be associated with.

Return Type

This method returns an object of Role that got created by CreateRole method.

Examples:

The following code creates a role named "TestRole_Company" associated with the Company ID "100001" and assigns it to the role object "outRole".

```
Role role = new Role();
role.Name = "TestRole_Company";
role.Description = "Testing CreateRole method with Company Association";
role.PreferredLanguage = SystemLanguages.English;
Analyzer2005 analyzerObject = InitializeAnalyzer();
Role outRole = analyzerObject.CreateRole(secContext, role, 100001); // you can also pass null for
securitycontext
```

4.3.3 UpdateRole Method

This method updates properties of a role based on the with the Role object passed in as parameter

Syntax:

```
UpdateRole(SecurityContext context, Role role)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
role
```

The role object having updated details of role that needs to be updated

Return Type

This method returns an object of Role.

Examples:

The following code creates a role named "TestRole" and updates the description of the same role.

```
Role role = new Role();
role.Name = "TestRole";
role.Description = "Testing UpdateRole method";
role.PreferredLanguage = SystemLanguages.English;
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Creating new Role
Role outRole = analyzerObject.CreateRole(secContext, role);

// Updating Role
outRole.Description = "Updating descripotion";
outRole.PreferredLanguage = SystemLanguages.SimpChinese;
```

```
outRole = analyzerObject.UpdateRole(secContext, outRole);
```

4.3.4 DeleteRole Method

This method deletes a specific role based on RoleId passed in as parameter

Syntax:

```
DeleteRole(SecurityContext context, int roleId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
roleId
```

The Id of the role that needs to be deleted

Return Type

This method returns an object of Role.

Examples:

The following code deletes the role 123654879.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
analyzerObject.DeleteRole(secContext, 123654879);
```

4.3.5 AddRoleMember Method

This method adds a specific member to a specific role.

Syntax:

```
AddRoleMember(SecurityContext context, int roleId, RoleMember member)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
roleId
```

The Id of the role that needs to be deleted

```
member
```

The RoleMember of the role that needs to be deleted

Return Type

This method returns an object of RoleMember that got created by CreateRoleMember method.

Examples:

The following code adds the member 123456789 to the role 114257893.

```
RoleMember roleMember = new RoleMember();
roleMember.MemberId = 123456789;
roleMember.MemberType = RoleMemberType.User;
Analyzer2005 analyzerObject = InitializeAnalyzer();
RoleMember outRoleMember = analyzerObject.AddRoleMember(secContext, 114257893, roleMember);
```

4.3.6 DeleteRoleMember Method

This method deletes a specific member from a specific role.

Syntax:

```
DeleteRoleMember(SecurityContext context, int roleId, RoleMember member)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
roleId
```

The Id of the role that needs to be deleted

```
member
```

The RoleMember of the role that needs to be deleted

Return Type

This method returns an object of RoleMember.

Examples:

The following code deletes the member 123456789 from the role 114257893.

```
RoleMember roleMember = new RoleMember();
roleMember.MemberId = 123456789;
Analyzer2005 analyzerObject = InitializeAnalyzer();
RoleMember outRoleMember = analyzerObject.DeleteRoleMember(secContext, 114257893, roleMember);
```

4.3.7 ClearRoleMembers Method

This method deletes all members from a specific role.

Syntax:

```
ClearRoleMembers(SecurityContext context, int roleId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
roleId
```

The Id of the role that needs to be deleted

Return Type

This method returns an object of BaseState class.

Examples:

The following code deletes all the members from the role 114257893.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
RoleMember outRoleMember = analyzerObject.ClearRoleMembers(secContext, 114257893);
```

4.3.8 ListRoles Method

This method retrieves the list of Roles that are present in the AnalyzerApplication instance for a specific security context.

Syntax:

```
ListRoles(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns collection of Role.

Examples:

The following code lists all the roles available in Analyzer application instance..

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
Role[] roles = analyzerObject.ListRoles(secContext);
```

4.4 User Management APIs

4.4.1 CreateUser Method

This method creates a new user based on the with the user object passed in as parameter.

Syntax:

```
CreateUser(SecurityContext context, User user)
```

Arguments:

```
context
```

The security context under which the CreateUser action needs to be performed.

```
user
```

The user object having the details of new user to be created

Return Type

This method returns an object of User that got created by CreateUser method.

Examples:

The following code creates a user named “TestUser” and returns the user object “outUser”.

```
User user = new User();
user.Name = “TestUser”;
user.Description = “Testing CreateUser API”;
user.DefaultHomePage = DefaultHomePages.PersonalReports;
user.EMailAddress = “testuser@testdomain.com”;
user.PreferredLanguage = SystemLanguages.English;
user.RowsPerPage = RowsPerPage.Fifteen;
user.AllowSendMessage = true;
```

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
User outUser = analyzerObject.CreateUser(secContext, user); // you can also pass null for securitycontext
```

4.4.2 CreateUser Method (Company specific)

This overloaded method creates a new user based on the user object passed in as parameter. This user profile is associated with a specific company based on the Company Id passed in as parameter.

Syntax:

```
CreateUser(SecurityContext context, User user , int companyId)
```

Arguments:

```
context
```

The security context under which the CreateUser action needs to be performed.

```
user
```

The user object having the details of new user to be created

```
companyId
```

The CompanyId which should be associated with the newly created user.

Return Type

This method returns an object of User that got created by CreateUser method.

Examples:

The following code creates a user named "TestUser_Company" associated with Company ID "100001" and returns the user object "outUser".

```
User user = new User();
user.Name = "TestUser_Company";
user.Description = "Testing CreateUser API with Company Assocaition";
user.DefaultHomePage = DefaultHomePages.PersonalReports;
```

```
   user.EMailAddress = "testuser_with_company@testdomain.com";
   user.PreferredLanguage = SystemLanguages.English;
   user.RowsPerPage = RowsPerPage.Fifteen;
   user.AllowSendMessage = true;

   Analyzer2005 analyzerObject = InitializeAnalyzer();
   User outUser = analyzerObject.CreateUser(secContext, user, 100001); // you can also pass null for
securitycontext
```

4.4.3 UpdateUser Method

This method updates properties of an userbased on the with the user object passed in as parameter

Syntax:

```
   UpdateUser(SecurityContext context, User user)
```

Arguments:

```
   context
```

The security context under which this action needs to be performed.

```
   user
```

The user object having updated details of role that needs to be updated

Return Type

This method returns an object of User.

Examples:

The following code creates a user named "TestUser" and updates default home page and rows per page properties of the same user.

```
   User user = new User();
```

```
user.Name = "TestUser";
user.Description = "Testing UpdateUser API";
user.DefaultHomePage = DefaultHomePages.PersonalReports;
user.EMailAddress = "testuser@testdomain.com";
user.PreferredLanguage = SystemLanguages.English;
user.RowsPerPage = RowsPerPage.Fifteen;
user.AllowSendMessage = true;

Analyzer2005 analyzerObject = InitializeAnalyzer();
// Creating new User
User outUser = analyzerObject.CreateUser(secContext, user); // you can also pass null for securitycontext

// Updating user
outUser.DefaultHomePage = DefaultHomePages.SharedReports;
outUser.RowsPerPage = RowsPerPage.Thirty;
outUser = analyzerObject.UpdateUser(secContext, outUser);
```

4.4.4 DeleteUser Method

This method deletes a specific user based on UserId passed in as parameter

Syntax:

```
DeleteUser(SecurityContext context, int userId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
userId
```

The Id of the user that needs to be deleted

Return Type

This method returns an object of User.

Examples:

The following code deletes the user 236547894.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
analyzerObject.DeleteUser(secContext, 236547894);
```

4.4.5 GetUserRoles Method

This method retrieves the list of Roles that for a specific User.

Syntax:

```
GetUserRoles(SecurityContext context, string userName)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
userName
```

The user name for which the Roles need to be listed.

Return Type

This method returns collection of Role object.

Examples:

The following code lists all the Roles that the user "TestUser" belongs to.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
```

```
Role[] roles = analyzerObject.GetUserRoles(secContext, "TestUser");
```

4.4.6 ListUsers Method

This method retrieves the list of Users that are present in the AnalyzerApplication instance for a specific security context.

Syntax:

```
ListUsers(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns collection of User.

Examples:

The following code lists all the users available in Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
User[] users = analyzerObject.ListUsers(secContext);
```

4.5 Company APIs

4.5.1 GetCompanies Method

This method retrieves a collection of Companies for the specific security context in the Analyzer system.

Syntax:

```
GetCompanies(SecurityContext context)
```

Arguments:

```
context
```

The security context under which the list of Companies has to be retrieved.

Return Type

This method returns an array of Company objects.

Examples:

The following code gets the list of all the users under the specifc security context.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Retrieve the list of all the Companies in the current security context
Company[] companies = analyzerObject.GetCompanies(secContext);
```

4.5.2 GetCompany Method

This method retrieves a Company object for the specific CompanyId.

Syntax:

```
GetCompany(SecurityContext context , int companyId)
```

Arguments:

```
context
```

The security context under which the Company details has to be retrieved.

```
CompanyId
```

The CompanyId for which the details have to be retrieved.

Return Type

This method returns an object of Company.

Examples:

The following code gets Company object for the CompanyId 100001.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Retrieve the Company object basd on a specific CompanyID
Company companies = analyzerObject.GetCompany(secContext, 100001);
```

4.5.3 GetCompanyFolder Method

This method retrieves a CatalogItem object with the folder details for the specific CompanyId.

Syntax:

```
GetCompanyFolder(SecurityContext context , int companyId)
```

Arguments:

```
context
```

The security context under which the desired action has to be performed.

```
CompanyId
```

The CompanyId for which the Folder details have to be retrieved.

Return Type

This method returns an object of CatalogItem.

Examples:

The following code gets CatalogItem object having the folder details for the CompanyId 100001.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Retrieve the Company Folder details basd on a specific CompanyID
CatalogItem companyFolder = analyzerObject.GetCompanyFolder(secContext, 100001);
```

4.5.4 GetRolesByCompanyId Method

This method retrieves a collection of Role objects for the specific CompanyId.

Syntax:

```
GetRolesByCompanyId(SecurityContext context , int companyId)
```

Arguments:

```
context
```

The security context under which the desired action has to be performed.

```
CompanyId
```

The CompanyId for which the list of Roles has to be retrieved.

Return Type

This method returns an array of Role objects.

Examples:

The following code gets collection of Role objects for the CompanyId 100001.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Retrieve the Roles for a specific Company based on the CompanyID
Role[] roles = analyzerObject.GetRolesByCompanyId(secContext, 100001);
```

4.5.5 GetUsersByCompanyId Method

This method retrieves a collection of User objects for the specific CompanyId.

Syntax:

```
GetUsersByCompanyId(SecurityContext context , int companyId)
```

Arguments:

```
context
```

The security context under which the desired action has to be performed.

```
CompanyId
```

The CompanyId for which the list of Users has to be retrieved.

Return Type

This method returns an array of User objects.

Examples:

The following code gets collection of User objects for the CompanyId 100001.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Retrieve the Users for a specific Company based on the CompanyID
User[] users = analyzerObject.GetUsersByCompanyId(secContext, 100001);
```

4.6 Folder Management APIs

4.6.1 CreateFolder Method

This method creates a folder under specified parent folder using the folder object passed as parameter.

Syntax:

```
CreateFolder(SecurityContext context, CatalogItem folder)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
folder
```

The folder object having the details of the new folder that needs to be created.

Return Type

This method returns a CatalogItem object.

Examples:

The following code creates a folder named "Test Folder" under "Shared Reports" folder.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem folder = new CatalogItem();
folder.ItemType = CatalogItemType.Folder;
folder.Name = "Test Folder";
folder.Description = "Testing CreateFolder API";
folder.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
                            // Refer to the specific section for the method FindCatalogItemByName
CatalogItem outFolder = analyzerObject.CreateFolder(secContext, folder);
```

4.6.2 UpdateFolder Method

This method updates a folder using the folder object passed as parameter.

Syntax:

```
UpdateFolder(SecurityContext context, CatalogItem folder)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
folder
```

The folder object having the details of the folder that needs to be updaed.

Return Type

This method returns a CatalogItem object.

Examples:

The following code creates a folder named "Test Folder" under "Shared Reports" folder and then updates the description of this folder.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem folder = new CatalogItem();
folder.ItemType = CatalogItemType.Folder;
folder.Name = "Test Folder";
folder.Description = "Testing UpdateFolder API";
folder.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
                            // Refer to the specific section for the method FindCatalogItemByName
// Creating new Folder
CatalogItem outFolder = analyzerObject.CreateFolder(secContext, folder);

// Updating Folder
```

```
folder.Description = "Testing UpdateFolder API";
outFolder = analyzerObject.UpdateFolder(secContext, folder);
```

4.6.3 DeleteFolder Method

This method deletes a specific folder based on folderId passed in as parameter

Syntax:

```
DeleteFolder(SecurityContext context, int folderId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
folderId
```

The Id of the folder that needs to be deleted

Return Type

This method returns an object of CatalogItem.

Examples:

The following code deletes the folder 223365476.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
analyzerObject.DeleteFolder(secContext, 223365476);
```

4.7 DataSource Management APIs

4.7.1 CreateDataSource Method

This method creates a datasource using the datasource object passed as parameter.

Syntax:

```
CreateDataSource(SecurityContext context, DataSource datasource)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
datasource
```

The datasource object having the details of the new datasource that needs to be created.

Return Type

This method returns a datasource object.

Examples:

The following code creates a folder named "Test Data Source" which points to the Analysis Server 2005 named "OLAPSVR1".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
DataSource dataSource = new DataSource();
dataSource.ServerName = "OLAPSVR1";
dataSource.Name = "Test Data Source";
dataSource.SourceType = DataSourceType.SqlAs2005;
dataSource.Description = "Testing CreateDataSource() API"
DataSource outDataSource = analyzerObject.CreateDataSource(secContext, dataSource);
```

4.7.2 UpdateDataSource Method

This method updates a DataSource using the datasource object passed as parameter.

Syntax:

```
UpdateDataSource(SecurityContext context, DataSource datasource)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
datasource
```

The datasource object having the details of the datasource that needs to be updaed.

Return Type

This method returns a DataSource object.

Examples:

The following code creates a folder named "Test Data Source" which points to the Analysis Server 2000 named "OLAPSVR1" and updates the source type to SQL2005.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
DataSource dataSource = new DataSource();
dataSource.ServerName = "OLAPSVR1";
dataSource.Name = "Test Data Source";
dataSource.SourceType = DataSourceType.SqlAs2000;
dataSource.Description = "Testing UpdateDataSource() API"
// Creating DataSource
DataSource outDataSource = analyzerObject.CreateDataSource(secContext, dataSource);

// Updating the DataSource
outDataSource.SourceType = DataSourceType.SqlAs2005;
```

```
outDataSource = analyzerObject.UpdateDataSource(secContext, dataSource);
```

4.7.3 DeleteDataSource Method

This method deletes a specific datasource based on dataSourceId passed in as parameter

Syntax:

```
DeleteDataSource(SecurityContext context, int dataSourceId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
dataSourceId
```

The Id of the datasource that needs to be deleted

Return Type

This method returns an object of DataSource.

Examples:

The following code deletes the folder 235645127.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
analyzerObject.DeleteDataSource(secContext, 235645127);
```

4.7.4 ListDataSources Method

This method retrieves the list of DataSources that are present in the AnalyzerApplication instance for a specific security context.

Syntax:

```
ListDataSources(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns collection of DataSource.

Examples:

The following code lists all the data sources available in Analyzer application instance..

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
DataSource[] dataSources = analyzerObject. ListDataSources(secContext);
```

4.8 Search Function APIs

4.8.1 FindCatalogItemById Method

This method searches for a CatalogItem based on the Id passed as parameter.

Syntax:

```
FindCatalogItemById(SecurityContext context, int id)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
id
```

The Id of the CatalogItem that is being searched.

Return Type

This method returns a CatalogItem object.

Examples:

The following code returns CatalogItem object for the Id 123456789.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem catalogItem = analyzerObject.FindCatalogItemById(secContext, 123456789);
```

4.8.2 FindCatalogItemByName Method

This method searches for a CatalogItems based on the name passed as parameter.

Syntax:

```
FindCatalogItemByName(SecurityContext context, string name)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
name
```

The name of the CatalogItem(s) that is/are being searched.

Return Type

This method returns a collection of CatalogItem objects.

Examples:

The following code returns CatalogItem object collection for the CatalogItems named “My Report”.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem[] catalogItems = analyzerObject.FindCatalogItemByName(secContext, “My Report”);
```

4.8.3 FindRoleById Method

This method searches for a Role based on the Role Id passed as parameter.

Syntax:

```
FindRoleById(SecurityContext context, int roleId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
roleId
```

The Id of the Role that is being searched.

Return Type

This method returns a Role object.

Examples:

The following code returns Role object for the Id 23568974.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
Role role = analyzerObject.FindRoleById(secContext, 23568974);
```

4.8.4 FindRoleByName Method

This method searches for a Role based on the role name passed as parameter.

Syntax:

```
FindRoleByName(SecurityContext context, string roleName)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
roleName
```

The name of the Role being searched.

Return Type

This method returns a Role object.

Examples:

The following code returns Role object for the Role named "TestRole".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
Role[] role = analyzerObject.FindRoleByName(secContext, "TestRole");
```

4.8.5 FindUserById Method

This method searches for a Role based on the User Id passed as parameter.

Syntax:

```
FindUserById(SecurityContext context, int userId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
userId
```

The Id of the User that is being searched.

Return Type

This method returns a User object.

Examples:

The following code returns User object for the Id 12235645.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
User user = analyzerObject.FindUserById(secContext, 12235645);
```

4.8.6 FindUserByName Method

This method searches for a User based on the user name passed as parameter.

Syntax:

```
FindUserByName(SecurityContext context, string userName)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
userName
```

The name of the User being searched.

Return Type

This method returns User object.

Examples:

The following code returns User object for the User named “TestUser”.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
User user = analyzerObject.FindUserByName(secContext, “TestUser”);
```

4.8.7 ListChildren Method

This method retrieves all the child elements for a CatalogItem based on the Id of the CatalogItem passed as parameter. The frequent usage of this will be for listing the child folders.

Syntax:

```
FindUserByName(SecurityContext context, int parentId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
parentId
```

The Id of the CatalogItem of which the child items need to be retrieved..

Return Type

This method returns collection of CatalogItem object.

Examples:

The following code returns collection of CatalogItems that are children of the folder “Shared Reports”.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
int parentId = analyzerObject.FindCatalogItemByName(secContext, “Shared Reports”)[0].Id;
CatalogItem[] childElements = analyzerObject.ListChildren(secContext, parentId);
```

4.9 ObjectPermission APIs

4.9.1 AddObjectPermission Method

This method registers the configured permission on an object based on the objectPermission passed as parameter.

Syntax:

```
AddObjectPermission(SecurityContext context, ObjectPermission objectPermission)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
objectPermission
```

The objectPermission object which needs to be registered/configured in Analyzer application.

Return Type

This method returns a BaseState object from which we can check if the action was successful or not.

Examples:

The following code configures the execute permission on the “Shared Reports” folder for the role “General User”.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
int sharedReportsId = analyzerObject.FindCatalogItemByName(secContext, “Shared Reports”)[0].Id;
int memberId = analyzerObject.FindRoleByName(secContext, “General User”).Id;

ObjectPermission objectPermission = new ObjectPermission();
objectPermission.ObjectId = sharedReportsId;
objectPermission.MemberId = memberId;
objectPermission.MemberType = SecurityMemberType.Role;
objectPermission.Permission = 16; // 16 is for Execute.
```

```
                                        // refer to the class ObjectPermission to know about more permissions
analyzerObject.AddObjectPermission(secContext, objectPermission);
```

4.9.2 ChangeObjectPermission Method

This method modify the permission configuration on an object based on the objectPermission passed as parameter.

Syntax:

```
ChangeObjectPermission(SecurityContext context, ObjectPermission objectPermission)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
objectPermission
```

The objectPermission object which needs to be configured in Analyzer application.

Return Type

This method returns a BaseState object from which we can check if the action was successful or not.

Examples:

The following code configures the execute permission on the "Shared Reports" folder for the role "General User" and then changes the permission to Execute and Export.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
int sharedReportsId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
int memberId = analyzerObject.FindRoleByName(secContext, "General User").Id;

ObjectPermission objectPermission = new ObjectPermission();
objectPermission.ObjectId = sharedReportsId;
objectPermission.MemberId = memberId;
```

```
objectPermission.MemberType = SecurityMemberType.Role;
objectPermission.Permission = 16; // 16 is for Execute.
                                  // refer to the class ObjectPermission to know about more permissions

// Adding object permission
analyzerObject.AddObjectPermission(secContext, objectPermission);

// Changing the object permission
objectPermission.Permission = 80; // 16 for Excute and 64 for Export. Hence 80 in Total
analyzerObject.ChangeObjectPermission(secContext, objectPermission);
```

4.9.3 RemoveObjectPermission Method

This method removes all the permissions configured on an object for a specific Member.

Syntax:

```
RemoveObjectPermission(SecurityContext context, int objectId, int memberId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
objectId
```

The Id of the object object on which the permission needs to be removed.

```
memberId
```

The Id of the Member for whom the permission should be removed.

Return Type

This method returns a BaseState object from which we can check if the action was successful or not.

Examples:

The following code removes all the permissions on the Folder "Shared Reports" for the Role "General User";

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
int sharedReportsId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
int memberId = analyzerObject.FindRoleByName(secContext, "General User").Id;
// Removing permissions on "Shared Reports" for "General User"
analyzerObject.ChangeObjectPermission(secContext, sharedReportsId, memberId);
```

4.9.4 ClearObjectPermissions Method

This method removes all the permissions configured on an object for all Members.

Syntax:

```
ClearObjectPermissions(SecurityContext context, int objectId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
objectId
```

The Id of the object object on which the permission needs to be removed.

Return Type

This method returns a BaseState object from which we can check if the action was successful or not.

Examples:

The following code removes all the permissions on the Folder "Shared Reports" for all the members having permission on this folder.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
int sharedReportsId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
// Removing permissions on "Shared Reports"
analyzerObject.ClearObjectPermissions(secContext, sharedReportsId);
```

4.9.5 GetObjectPermissions Method

This method retrieves all the permissions configured on a CatalogItem. If a CatalogItem has ***X*** number of members having different permissions, then this method will return collection of ObjectPermission objects of size ***X***.

Syntax:

```
GetObjectPermissions(SecurityContext context, CatalogItem item)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
item
```

The CatalogItem object for which the permissions need to be retrieved.

Return Type

This method returns a collection of ObjectPermission object.

Examples:

The following code retrieves all the ObjectPermission on the Folder "Shared Reports" for all the members.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem sharedReports = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports");

ObjectPermission[] sharedReportPermissions = analyzerObject.GetObjectPermissions(secContext, sharedReports);
```

4.9.6 GetUserPermission Method

This method retrieves all the permissions configured on a CatalogItem for a specific user.

Syntax:

```
GetUserPermission(SecurityContext context, string userName, CatalogItem item)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
userName
```

The user name for whome the permissions need to be retrieved.

```
item
```

The CatalogItem object for which the permissions need to be retrieved.

Return Type

This method returns a PermissionInfo object.

Examples:

The following code retrieves the permissions set on the Folder "Shared Reports" for the role "TestUser".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem sharedReports = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports");

PermissionInfo userPermission = analyzerObject.GetObjectPermissions(secContext, "TestUser", sharedReports);
```

4.9.7 InheritParentPermissions Method

This method clones the permission on a Catalot Item from its parent.

Syntax:

```
InheritParentPermissions(SecurityContext context, CatalogItem item, bool inherited)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
item
```

The CatalogItem object on which the permission needs to be configures from its parent.

```
inherited
```

The flag to spcify whether to inherit the permission from the parent or not.

Return Type

This method returns a BaseState object indicating whether the action was successful or not.

Examples:

The following code creates a Folder named "Test Folder" under "Shared Reports" and sets the permission same as "Shared Reports".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem folder = new CatalogItem();
folder.ItemType = CatalogItemType.Folder;
folder.Name = "Test Folder";
folder.Description = "Testing InheritParentPermissions API";
folder.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
                              // Refer to the specific section for the method FindCatalogItemByName
// Creating "Test Folder"
CatalogItem outFolder = analyzerObject.CreateFolder(secContext, folder);

// Inherits permission from the parent folder
analyzerObject.InheritParentPermissions(secContext, outFolder, true);
```

4.9.8 IsDescendFrom Method

This method indicates whether a CatalogItem is descendant of a specific object based on the object Id which is passed as parameter.

Syntax:

```
IsDescendFrom(SecurityContext context, CatalogItem item, int ascendantId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
item
```

The CatalogItem object on which the permission needs to be checked whether it is descendant or not.

```
ascendantId
```

The object Id on which the permission sneeds to be checked whether it is ascendant or not.

Return Type

This method returns Boolean value indicating whether the CatalogItem is descendant or not.

Examples:

The following code shows how to find whether the folder "Test Folder" is descendant of the folder "Shared Reports". In this case the answer returned is true.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem folder = new CatalogItem();
int parentFolderId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;
folder.ItemType = CatalogItemType.Folder;
folder.Name = "Test Folder";
folder.Description = "Testing InheritParentPermissions API";
folder.ParentId = parentFolderId;
                              // Refer to the specific section for the method FindCatalogItemByName
// Creating "Test Folder"
CatalogItem outFolder = analyzerObject.CreateFolder(secContext, folder);

// To check whether "Test Folder" is descendant of "Shared Folder"
// The value of isDescendant after this method call will be true.
bool isDescendant = analyzerObject. IsDescendFrom(secContext, outFolder, parentFolderId);
```

4.10 System Administration APIs

4.10.1 ListSystemLogs Method

This method retrieves all the system log entries in Analyzer application instance. We can generate system log report using the collection of SystemLog returned from this method.

Syntax:

```
ListSystemLogs(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns a collection of SystemLog objects.

Examples:

The following code shows how to retrieve the system log entries from Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SystemLog[] systemLogCollection = analyzerObject.ListSystemLogs(secContext);
```

4.10.2 BackupSystemLog Method

This method creates a back up for the current system log entries. This is often used as archival action when the system log is being cleared.

Syntax:

```
BackupSystemLog(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns a SystemLog object.

Examples:

The following code shows how to back up system log in Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SystemLog systemLogBackup = analyzerObject. BackupSystemLog(secContext);
```

4.10.3 TruncateSystemLog Method

This method clears the current system log entries in Analyzer application instance.

Syntax:

```
TruncateSystemLog(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns a SystemLog object.

Examples:

The following code shows how to clean up system log in Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
SystemLog systemLogBackup = analyzerObject. TruncateSystemLog(secContext);
```

4.10.4 ChangeSystemLogPolicy Method

This method clears the current system log entries in Analyzer application instance.

Syntax:

```
ChangeSystemLogPolicy(SecurityContext context, LogPolicy policy)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
policy
```

The LogPolicy object having the updated policy information.

Return Type

This method returns a BaseState object from which we can check the status of this method.

Examples:

The following code shows how to update the system log policy in Analyzer application instance. The configurations that are being changed are BackupBeforeTruncate flag and the maximum number of records after which the Truncate action has to be performed.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
LogPolicy updatedLogPolicy = new LogPolicy();
// set the system to backup the logs before truncating
updatedLogPolicy.BackupBeforeTruncate = true;
// set the maximum number of records after which the system log should be truncated
updatedLogPolicy.MaxRecords = 10000;

// Update the log policy
analyzerObject.ChangeSystemLogPolicy(secContext, updatedLogPolicy);
```

4.10.5 ConfigMailServer Method

This method clears the current system log entries in Analyzer application instance.

Syntax:

```
ConfigMailServer(SecurityContext context, string serverName, int port, string userId, string password,
                        string defaultSender, int authenticate, int sendUsing, int useSSL)
```

Arguments:

`context`

The security context under which this action needs to be performed.

`serverName`

The SMTP server which can be used for sending e-mail.

`port`

The port number that can be used for sending e-mail.

`userId`

The user Id having previleges to access the SMTP server for sending e-mails. It's usually the e-mail id.

`password`

The password of the User specified for the userId parameter.

`defaultSender`

This should be a valid e-mail Id which will be used for sending e-mails and this e-mail address will be in the "*From*" section of the e-mail.

`authenticate`

You can pass the value 1 for this parameter.

`sendUsing`

You can pass the value 2 for this parameter.

`useSSL`

You can pass the value 0 for this parameter.

Please refer to the section E-mail Configuration for more information on the above mentioned parameters.

Return Type

This method returns a BaseState object from which we can check the status of this method.

Examples:

The following code shows how to update the mail server configuration in Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Update the Mail Server Configuration
analyzerObject. ConfigMailServer(secContext, mailserver.testdomain.com, 25, "testuser", "testpassword",
                        testuser@testdomain.com, authenticate, sendUsing, useSSL);
```

4.11 Report Instance Management APIs

4.11.1 CreateReport Method

This method creates a new report with the report details passed in as parameter. The CatalogItem of type report will be passed into this method.

Syntax:

```
CreateReport(SecurityContext context, CatalogItem report, bool save)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
report
```

The CatalogItem object of type report having the details of the report to be created.

```
save
```

The Boolean value indicating whether to save the report into Analyzer application by default when the report is created.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to create a new empty report named "First Report" which is not associated with any cube/database/datasource. This code will return an object of ReportInstance which gets saved in Analyzer application. If you login to the Analyzer application in Stand Along mode after this method call, you can see this new report under Shared Reports and when you try to execute this report, the report page will be displayed but no cube associated with this report.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
```

```
report.Name = "First Report";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Create empty report. The last parameter is set as true so that the report will be saved.
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, true);
```

4.11.2 CreateReportWithOptions Method

This method creates a new report predefined set of cube associated with it along with the custom UI preferences. Using this method, you can decide which cube should be associated with the report and what are all the buttons that will be displayed (or hidden) toolbar on the user interface. You can also specify whether to display the datasource panel (which allows users to browse through the datasource/database/cubes and add them to the report) or the schema (which allows user to drag and drop dimensions and measures to customize the report).

Syntax:

```
CreateReportWithOptions(SecurityContext context, CatalogItem report, int dataSourceId, string databaseName,
                                    string cubeName, PivotTable pivotTable, ToolbarButton[] hiddenButtons,
                                    bool schemaVisible, bool dataSourceVisible)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
report
```

The CatalogItem object of type report having the details of the report to be created.

```
dataSourceId
```

The dataSourceId to specify the Analysis Server that contains the OLPA database.

```
databaseName
```

The OLPA database that contains the cube.

```
cubeName
```

The cube Name which needs to be associated with the report.

`pivotTable`

The pivotTable object, which will be part of the new report that is being created.

`hiddenButtons`

The collection of toolbarButtons which needs to be hidden on the user interface.

`schemaVisible`

Boolean value indicating whether to display the schema of the cube for allowing users to drag and drop dimensions and measures.

`dataSourceVisible`

Boolean value indicating whether to display the DataSource panel on the user interface allowing user to access different datasource/database/cubes to design the report.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to create a new report named "Report With cubes" which is associated with any "Sales" cube under "Adventure Works DW" on the OLAP server "OLAPSVR1" (Added as a DataSource with the name "Test Data Source"). The buttons SchemaTree, DataSource and Close are being hidden by passing a collection of toolbar buttons as parameter. Also the panels DataSource is being hidden on the user interface but the Schema panel is being displayed so that the user can drag and drop the dimensions and measures to the report.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
```

```
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;

PivotTable pivotTable = new PivotTable();

// Configure the toolbar. Create a collection of toolbarbuttons that need to be hidden in the UI
ToolbarButton[] buttons = new ToolbarButton[] {
                              ToolbarButton.DataSource,
                              ToolbarButton.SchemaTree,
                              ToolbarButton.Close};


ReportInstance reportInstance = analyzerObject.CreateReportWithOptions
                                                (secContext,
                                                report,
                                                dataSourceId,
                                                "Adventure Works DW",
                                                "Sales",
                                                pivotTable,
                                                buttons,
                                                true,       // Setting Schem panel visible
                                                false);     // Hiding DataSource panel
```

4.11.3 CreateReportWithOptions2 Method

This meathod is exactly similar to the method CreateReportWithOptions but with an extended feature of additional cubes to the report which can be used for generating report.

Syntax:

```
CreateReportWithOptions2(SecurityContext context, CatalogItem report, int dataSourceId,
                                    string databaseName, string cubeName, PivotTable pivotTable,
                                    AdditionalCube[] cubes, ToolbarButton[] hiddenButtons,
                                    bool schemaVisible, bool dataSourceVisible)
```

Arguments:

`context`

The security context under which this action needs to be performed.

`report`

The CatalogItem object of type report having the details of the report to be created.

`dataSourceId`

The dataSourceId to specify the Analysis Server that contains the OLPA database.

`databaseName`

The OLPA database that contains the cube.

`cubeName`

The cube Name which needs to be associated with the report.

`pivotTable`

The pivotTable object, which will be part of the new report that is being created.

`hiddenButtons`

The collection of toolbarButtons which needs to be hidden on the user interface.

`schemaVisible`

Boolean value indicating whether to display the schema of the cube for allowing users to drag and drop dimensions and measures.

`dataSourceVisible`

Boolean value indicating whether to display the DataSource panel on the user interface allowing user to access different datasource/database/cubes to design the report.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to create a new report named "Report With additional cubes" which is associated with any "Sales" cube under "Adventure Works DW" on the OLAP server "OLAPSVR1" (Added as a DataSource with the name "Test Data Source"). It also has two additional cubes "Channel Sales" and "Direct Sales" under the same database as of the main cube associated with the report. The buttons SchemaTree, DataSource and Close are being hidden by passing a collection of toolbar buttons as parameter. Also the panels DataSource is being hidden on the user interface but the Schema panel is being displayed so that the user can drag and drop the dimensions and measures to the report.

```
// Create additional cube object based on the cube and database details
private AdditionalCube CreateAdditionalCube(int dataSourceId, string databaseName, string cubeName)
{
      AdditionalCube additionalCube = new AdditionalCube();
      additionalCube.DataSourceId = dataSourceId;
      additionalCube.DatabaseName = databaseName;
      additionalCube.CubeName = cubeName;

      return additionalCube;
}

Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with additional Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
// The test data source refers to the server "OLAPSVR1"
```

```
// This Analysis Server has the Adventure Works DW database.
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;

PivotTable pivotTable = new PivotTable();

// Creating a collection of Additional cubes that can be associated with the report
AdditionalCube[] cubes = new AdditionalCube[]
                              {
                                 CreateAdditionalCube(dataSourceId, "Adventure Works DW", "Channel Sales"),
                                 CreateAdditionalCube(dataSourceId, "Adventure Works DW", "Direct Sales")
                              };

// Configure the toolbar. Create a collection of toolbarbuttons that need to be hidden in the UI
ToolbarButton[] buttons = new ToolbarButton[] {
                              ToolbarButton.DataSource,
                              ToolbarButton.SchemaTree,
                              ToolbarButton.Close};

ReportInstance reportInstance = analyzerObject.CreateReportWithOptions2
                                                  (secContext,
                                                  report,
                                                  dataSourceId,
                                                  "Adventure Works DW",
                                                  "Sales",
                                                  pivotTable,
                                                  buttons,
                                                  true,       // Setting Schem panel visible
                                                  false);     // Hiding DataSource panel
```

4.11.4 OpenReport Method

This method creates helps in retrieving the report which has been saved in Analyzer application instance. Once the ReportInstance is available then this can be used to display the report on the user interface by using the other APIs available.

Syntax:

```
OpenReport(SecurityContext context, int reportId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
reportId
```

The Id of the report for which the ReportInstance needs to be genreated.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to obtain ReportInstance of a report with the Id 125469878 that has been saved in Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Open the report with the Id "125469878"
ReportInstance reportInstance = analyzerObject.OpenReport(secContext, 125469878);
```

4.11.5 OpenReportWithOptions Method

This method returns ReportInstance with the customized user interface configuration based on the configuration passed in as parameters. This method can be used if you need to display the saved report with custom toolbar buttons or hide/show some of the sections on the user interface.

Syntax:

```
OpenReportWithOptions(SecurityContext context, int reportId, ToolbarButton[] hiddenButtons,
                                    bool schemaVisible, bool dataSourceVisible)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
reportId
```

The Id of the report for which the ReportInstance needs to be generated.

```
hiddenButtons
```

The collection of toolbarButtons which needs to be hidden on the user interface.

```
schemaVisible
```

Boolean value indicating whether to display the schema of the cube for allowing users to drag and drop dimensions and measures.

```
dataSourceVisible
```

Boolean value indicating whether to display the DataSource panel on the user interface allowing user to access different datasource/database/cubes to design the report.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to create an instance of report which has the toolbar buttons DataSource, SchemaTree and Close hidden. Also the DataSource panel is hidden and Schema Panel is set to visible.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Configure the toolbar. Create a collection of toolbarbuttons that need to be hidden in the UI
ToolbarButton[] buttons = new ToolbarButton[] {
                            ToolbarButton.DataSource,
                            ToolbarButton.SchemaTree,
```

```
                                    ToolbarButton.Close};
ReportInstance reportInstance = analyzerObject.OpenReportWithOptions(secContext, 125469878,
                                                                     buttons, true, false);
```

4.11.6 OpenReportWithOptions2 Method

This method similar to the OpenReportWithOptions method which returns ReportInstance with the customized user interface configuration based on the configuration passed in as parameters. The only difference being you can pass a filter expression (MDX) which will be applied on the filter before returning the report instance.

Syntax:

```
OpenReportWithOptions2(SecurityContext context, int reportId, ToolbarButton[] hiddenButtons,
                                        bool schemaVisible, bool dataSourceVisible, string[] reportFilter)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
reportId
```

The Id of the report for which the ReportInstance needs to be generated.

```
hiddenButtons
```

The collection of toolbarButtons which needs to be hidden on the user interface.

```
schemaVisible
```

Boolean value indicating whether to display the schema of the cube for allowing users to drag and drop dimensions and measures.

```
dataSourceVisible
```

Boolean value indicating whether to display the DataSource panel on the user interface allowing user to access different datasource/database/cubes to design the report.

```
reportFilter
```

A string collecton of MDX filter expressions which will be applied on the report output and then return the report instance with the applied filter.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to create an instance of report which has the toolbar buttons DataSource, SchemaTree and Close hidden. Also the DataSource panel is hidden and Schema Panel is set to visible.

```
   Analyzer2005 analyzerObject = InitializeAnalyzer();

   string[] reportFilter = null;

// Configure the toolbar. Create a collection of toolbarbuttons that need to be hidden in the UI
   ToolbarButton[] buttons = new ToolbarButton[] {
                                ToolbarButton.DataSource,
                                ToolbarButton.SchemaTree,
                                ToolbarButton.Close};
   ReportInstance reportInstance = analyzerObject.OpenReportWithOptions2(secContext, 125469878,
                                                                         buttons, true, false, reportFilter);
```

4.11.7 OpenReportWithOptions3 Method

This method is similar to the above two methods with an additional option of configuring the connection timout related parameter through ConnectionParameters object.

Syntax:

```
   OpenReportWithOptions3(SecurityContext context, int reportId, ToolbarButton[] hiddenButtons,
   bool schemaVisible, bool dataSourceVisible, string[] reportFilter, ConnectionParameters connectionParams)
```

Arguments:

```
   context
```

The security context under which this action needs to be performed.

`reportId`

The Id of the report for which the ReportInstance needs to be generated.

`hiddenButtons`

The collection of toolbarButtons which needs to be hidden on the user interface.

`schemaVisible`

Boolean value indicating whether to display the schema of the cube for allowing users to drag and drop dimensions and measures.

`dataSourceVisible`

Boolean value indicating whether to display the DataSource panel on the user interface allowing user to access different datasource/database/cubes to design the report.

`reportFilter`

A string collecton of MDX filter expressions which will be applied on the report output and then return the report instance with the applied filter.

`connectionParams`

An oject of ConnectionParameters class configured with the timout options for the report.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to create an instance of report which has the toolbar buttons DataSource, SchemaTree and Close hidden. Also the DataSource panel is hidden and Schema Panel is set to visible. Also configured is the connection parameters where the ConnectTimeout is set to 499 seconds, Timeout is set to 399 secons and the custom data is “UserA”.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

string[] reportFilter = null;
// Configure the connection parameters
ConnectionParameters connectionParams = new ConnectionParameters();
connectionParams.ConnectTimeout = 499;
```

```
connectionParams.Timeout = 399;
connectionParams.CustomData = "UserA";
// Configure the toolbar. Create a collection of toolbarbuttons that need to be hidden in the UI
ToolbarButton[] buttons = new ToolbarButton[] {
                                ToolbarButton.DataSource,
                                ToolbarButton.SchemaTree,
                                ToolbarButton.Close};
ReportInstance reportInstance = analyzerObject.OpenReportWithOptions3(secContext, 125469878,
                                                        buttons, true, false, reportFilter, connectionParams);
```

4.11.8 GetReportInstanceUrl Method

This method generates to URL for a specific report which can be used to display the report to the user on Web Browser.

Syntax:

```
GetReportInstanceUrl(SecurityContext context, ReportInstance instance, string serverPath)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The report instance object for which the report URL needs to be generated.

```
serverPath
```

The application path where Analyzer has been installed.

Return Type

This method returns string which has the URL for the report.

Examples:

The following code shows how to obtain URL for a given ReportInstance of the report with the Id 125469878.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Open the report with the Id "125469878"
ReportInstance reportInstance = analyzerObject.OpenReport(secContext, 125469878);

// "MACHINE1.DOMAIN" is the web server where Analyzer application has been installed
// where Virtual Directory Name is "Analyzer"
string reportUrl = analyzerObject.GetReportInstanceUrl(secContext, reportInstance,
                                                        "MACHINE1.DOMAIN/Analyzer");
```

4.11.9 SaveReport Method

This method performs save a Report into Analyzer application for the reportInstance passed in as parameter.

Syntax:

```
SaveReport(SecurityContext context, ReportInstance instance)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The report instance object having the report details which needs to be saved into Analyzer appliction.

Return Type

This method returns updated ReportInstance object.

Examples:

The following code gets an object of ReportInstance for the report with the Id 125469878 and then Save the report after the report has been modified.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Open the report with the Id "125469878"
ReportInstance reportInstance = analyzerObject.OpenReport(secContext, 125469878);

// do modifications on the report.... might be through user interface
// or you can use APIs to modify the reportInstance object

// Save the updated reportInstance into Analyzer
reportInstance = analyzerObject.SaveReport(secContext, reportInstance);
```

4.11.10SaveReportAs Method

This method performs copies a Report as a new report into Analyzer application for the reportInstance passed in as parameter. In this case the original report remain intact and any changes made will reflect on the newly saved report.

Syntax:

```
SaveReport(SecurityContext context, ReportInstance instance , int parentId, string newName)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The report instance object having the report details which needs to be saved into Analyzer appliction.

```
parentId
```

The Id of the folder under which a copy of the report need to be saved.

```
newName
```

The name for the new report which will be saved.

Return Type

This method returns updated ReportInstance object.

Examples:

The following code gets an object of ReportInstance for the report with the Id 125469878 and then Save the report under "Test Folder" with the name "New Report".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Open the report with the Id "125469878"
ReportInstance reportInstance = analyzerObject.OpenReport(secContext, 125469878);

// Get the Id of the folder "Test Folder"
int parentId = analyzerObject.FindCatalogItemByName(secContext, "Test Folder")[0].Id;

// Save the report under "Test Folder" with the name "New Report"
ReportInstance  newReportInstance = analyzerObject.SaveReportAs(secContext, reportInstance,
                                                                parentId, "New Report");
```

4.11.11 CloseReport Method

This method closes a specific instance of report. You can opt to save the report before closing it.

Syntax:

```
SaveReport(SecurityContext context, ReportInstance instance ,  bool saveBeforeClose)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The report instance object having the report details which needs to be saved into Analyzer appliction.

```
saveBeforeClose
```

Boolean value indicating whether to save the report before closing it or not.

Return Type

This method returns updated ReportInstance object.

Examples:

The following code gets an object of ReportInstance for the report with the Id 125469878 and then closes the instance of the report. It also saves the report before closing.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Open the report with the Id "125469878"
ReportInstance reportInstance = analyzerObject.OpenReport(secContext, 125469878);

// do modifications on the report
// Close the report instance and save it before closing
reportInstance = analyzerObject.CloseReport(secContext, reportInstance, true);
```

4.11.12 CopyReport Method

This method clones a specific report and returns the new report object as a CatalogItem with the new Owner and Folder details based on the parameters passed.

Syntax:

```
CopyReport(SecurityContext context, int reportId, int destFolderId, int newOwnerId)
```

Arguments:

```
context
```

The security context under which the report has to be copied into a new report.

```
reportId.
```

The report Id of the report which needs to be cloned into a new report.

```
destFolderId
```

The folder id where the newly cloned report needs to be saved in Analyzer.

```
newOwnerId
```

The owner (user) Id for the newly cloned report.

Return Type

This method returns CatalogItem object.

Examples:

The following code gets an object of CatalogItem for the new report which is cloned using the report with the Id 125469878.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Copy the report with the Id "125469878" into a new Report
CatalogItem item = analyzerObject.CopyReport(secContext, 125469878, 223365476,
                                    analyzerObject.FindUserByName(secContext, "DOMAIN/USER").Id);
```

4.12 Report Metadata Management APIs

4.12.1 DeleteReport Method

This method deletes a report based on the report Id passed as parameter. It is not required to create an instance of a report in this case, as just the metadata update will do the purpose.

Syntax:

```
CreateReport(SecurityContext context, int reportId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
reportId
```

Id of the report that has to be deleted.

Return Type

This method returns CatalogItem object.

Examples:

The following code deletes the report with the Id 125469878.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Delete the report with the Id "125469878"
CatalogItem deletedReport = analyzerObject.DeleteReport(secContext, 125469878);
```

4.12.2 UpdateReport Method

This method updates the metadata details of a report based on the report details passed as parameter. It is not required to create an instance of a report in this case, as just the CatalotItem object having the report details will do the purpose.

Syntax:

```
CreateReport(SecurityContext context, CatalogItem report)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
report
```

The CatalotItem object having modified details of the report which needs to be updated.

Return Type

This method returns CatalogItem object.

Examples:

The following code updates the metadata details of the report "First Report". It modifies the name, description and sets the parent folder to "Test Folder".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// retrieve the report object that needs to be modified
CatalogItem report = analyzerObject. FindCatalogItemByName(secContext, "First Report");

// Modify the report properties
report.Name = "First Modified Report";
report.Description = "Testing Update Report API";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Test Folder")[0].Id; ;
```

```
// update the report in Analyzer application
report = analyzerObject.UpdateReport(secContext, report);
```

4.12.3 GetReportLinkUrl Method

This method returns the URL for a saved report in Analyzer application. This method is exactly similar to the method GetReportInstanceUrl which was explained earlier in the Report Instance Management section. The difference is the reportId is enough to get the URL here as compared to the GetReportInstanceURL method where it requires an object of ReportInstance to retieve the URL. If you are working with a report and have the ReportInstance object ready, then you can invoke GetReportInstanceUrl method. But if you just have the report Id and want to retrieve the url through which the report can be accessed, then you need to invoke the method GetReportLinkUrl.

Syntax:

```
CreateReport(SecurityContext context, int reportId, string serverPath)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
reportId
```

Id of the report for which the report link Url needs to be retrieved.

```
serverPath
```

The application path where Analyzer has been installed.

Return Type

This method returns string which has the URL for the report.

Examples:

The following code shows how to obtain URL for a given report with the Id 125469878 on the machine "MACHINE1.DOMAIN".

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// "MACHINE1.DOMAIN" is the web server where Analyzer application has been installed
// where Virtual Directory Name is "Analyzer"
string reportUrl = analyzerObject.GetReportLinkUrl(secContext, 125469878,
                                                    "MACHINE1.DOMAIN/Analyzer");
```

4.13 User Interface Management APIs

4.13.1 InitializeCube Method

This method is used while building a cube report by using the API CreateReport where the CreateReport will just create an empty report. Using InitializeCube method you can update the cube information that the report is associated with. If you are using CreateReportWithOptions or CreateReportWithOptions2, you need not use this method because the association of report with the cube component will be taken care automatically.

Syntax:

```
InitializeCube(SecurityContext context, ReportInstance instance, string componentName, int dataSourceId,
                    string databaseName, string cubeName)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The ReportInstance object for which the cube needs to be initialized.

```
componentName
```

The name of the first PivtoTable in the report.

```
dataSourceId
```

Id of the datasource where the cube database resides.

```
databaseName
```

Name of the Database which contains the cube.

```
cubeName
```

Name of the cube which needs to be initialized.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and Initialize cube option.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.

// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");
```

4.13.2 InitializePivotTable Method

This method is used while building a cube report by using the API CreateReport where the CreateReport will just create an empty report. Using InitializePivotTable method, you can update the Pivot Table information (like the row/column/slicer dimensions and measures of the Pivot Table) that the report is associated with. If you are using CreateReportWithOptions or CreateReportWithOptions2, you will not be able to accomplish this task hence this method is used to add dimensions and measures to the pivot table of a specific report..

Syntax:

```
InitializePivotTable(SecurityContext context, ReportInstance instance, string componentName,
                                                                        PivotTable pivotTable)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The ReportInstance object for which the cube needs to be initialized.

```
componentName
```

The name of the first Pivot Table in the report.

```
pivotTable
```

PivotTable object which needs to be initialized.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and Initialize cube and Initialize Pivot Table option.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
```

```
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.



// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");

PivotTable pivotTable = new PivotTable();

// Set the row dimension for the pivot table
pivotTable.Rows = new Dimension[]
      {
            CreateDimension("Product", DimensionTypes.Hierarchy)
      };
```

```
// Set the column dimension for the pivot table
pivotTable.Columns = new Dimension[]
      {
            CreateDimension("Product", DimensionTypes.Hierarchy)
      };

// Set the slicer/filter dimension for the pivot table
pivotTable.Slicer = new Dimension[]
       {
            CreateDimension("Order Date", DimensionTypes.Hierarchy)
       };

 // Set the measure for the pivot table
 pivotTable.Measures = new Measure[]
       {
            CreateMeasure("Order Qty", MeasureTypes.Regular)
       };

 // Initialize the PivotTable object with the above configuration...
reportInstance = analyzerObject.InitializePivotTable(secContext, reportInstance,
                                                      pivotTableName, pivotTable);
```

4.13.3 AddAdditionalCube Method

This method is used to add additional cubes to the existing report. You can create and empty report with a default cube and then add additional cubes to it.

Syntax:

```
AddAdditionalCubes(SecurityContext context, ReportInstance instance, AdditionalCube[] cubes)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The ReportInstance object for which additional cubes need to be added.

```
cubes
```

The collection of cubes having the additional cube details.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and add additional cubes to it.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.
```

```
// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");

// Creating a collection of Additional cubes that can be added to the report
AdditionalCube[] additionalCubes = new AdditionalCube[]
      {
            // Refer to the earlier methods for creating the additional cube
            CreateAdditionalCube(dataSourceId, "Adventure Works DW", "Channel Sales"),
            CreateAdditionalCube(dataSourceId, "Adventure Works DW", "Direct Sales")
      };

// Add additional cubes ("Channel Sales" and "Direct Sales") to the report.
reportInstance = analyzerObject.AddAdditionalCubes(secContext, reportInstance, additionalCubes);
```

4.13.4 ClearAdditionalCube Method

This method is used to delete/clear all additional cubes in a report. This method will clear only the additional cubes leaving the default cube still associated with the report.

Syntax:

```
AddAdditionalCubes(SecurityContext context, ReportInstance instance)
```

Arguments:

`context`

The security context under which this action needs to be performed.

`instance`

The ReportInstance object from which additional cubes need to be deleted/cleared.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and clear the additional cubes added to it.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.

// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
```

```
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");

// Creating a collection of Additional cubes that can be added to the report
AdditionalCube[] additionalCubes = new AdditionalCube[]
      {
            // Refer to the earlier methods for creating the additional cube
            CreateAdditionalCube(dataSourceId, "Adventure Works DW", "Channel Sales"),
            CreateAdditionalCube(dataSourceId, "Adventure Works DW", "Direct Sales")
      };

// Add additional cubes ("Channel Sales" and "Direct Sales") to the report.
reportInstance = analyzerObject.AddAdditionalCubes(secContext, reportInstance, additionalCubes);

// Clear all the additional cubes.
// This method will clear "Channel Sales" and "Direct Sales" cubes but not "Sales"
reportInstance = analyzerObject.ClearAdditionalCubes(secContext, reportInstance);
```

4.13.5 AddCustomToolbarButtons Method

This method is used to add custom toolbar buttons to the existing report. On click of the custom toolbar button you can trigger your own code. You should have the javascript which is accessible by the windw/frame where the Analyzer report is being loaded. If you have a web page WebPage1 and an iFrame AnalyzerFrame in this web page, then the WebPage1 have the javascript file having the function that you want to invoke. You can also include the javascript file having this function into WebPage1.

Syntax:

```
AddCustomToolbarButtons(SecurityContext context, ReportInstance instance, CustomButton[] buttons)
```

Arguments:

`context`

The security context under which this action needs to be performed.

`instance`

The ReportInstance object for which custom toolbar buttons need to be added.

`buttons`

The collection of custom toolbar buttons.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and add custom toolbar buttons to it.

```
// The following method can be used for creating custom button and
// use it while adding custom toolbar button to the report
private static CustomButton CreateButton(string name, string hint, string icon, string script)
     {
         CustomButton customButton = new CustomButton();
         customButton.Name = name;
         customButton.Hint = hint;
         customButton.Icon = icon;
         customButton.Script = script;
         return customButton;
     }

////////////////////////////////////////////////////////////////////////////
Analyzer2005 analyzerObject = InitializeAnalyzer();
```

```
// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.

// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");

// Creating a collection of custom toolbar buttons that can be added to the report
CustomButton[] customToolbarButtons = new CustomButton[]
      {
            // You can specify any javascript function that will be accessible by the window
            // where this report is being rendered
            CreateButton("btnTest", "Test custom toolbar button",
```

```
            "http://WEBSERVER/Images/customButton.gif", "alert(\"You have clicked on custom button\")")
        };

// Add custom toolbar button to the report.
reportInstance = analyzerObject.AddCustomToolbarButtons(secContext, reportInstance, customToolbarButtons);
```

4.13.6 ClearCustomToolbarButtons Method

This method is used to clear custom toolbar buttons in a report. This method will not affect any of the Analyzer default button which are hidden.

Syntax:

```
ClearCustomToolbarButtons(SecurityContext context, ReportInstance instance)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The ReportInstance object for which custom toolbar buttons need to be added.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and clear custom toolbar buttons added to it.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Create a report object
CatalogItem report = new CatalogItem();
```

```
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.

// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");

// Creating a collection of custom toolbar buttons that can be added to the report
CustomButton[] customToolbarButtons = new CustomButton[]
      {
            // You can specify any javascript function that will be accessible by the window
            // where this report is being rendered
            CreateButton("btnTest", "Test custom toolbar button",
            "http://WEBSERVER/Images/customButton.gif", "alert(\"You have clicked on custom button\")")
      };
```

```
// Add custom toolbar button to the report.
reportInstance = analyzerObject.AddCustomToolbarButtons(secContext, reportInstance, customToolbarButtons);


// Clear all the custom toolbar buttons
reportInstance = analyzerObject.ClearCustomToolbarButtons(secContext, reportInstance);
```

4.13.7 HideToolbarButtons Method

This method is used to hide specifc toolbar buttons (standard buttons). For example if you do not want the users to see the Save button, or schemaTree button, then you might achieve this by using HideToolbarButtons method.

Syntax:

```
HideToolbarButtons(SecurityContext context, ReportInstance instance, ToolbarButton[] buttons)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
instance
```

The ReportInstance object for which toolbar buttons need to be hidden.

```
buttons
```

Collection of toolbar buttons that need to be hidden.

Return Type

This method returns ReportInstance object.

Examples:

The following code shows how to build a report using CreateReport and hide the toolbar buttons “SchemaTree”, “DataSource” and “Save”.

```


Analyzer2005 analyzerObject = InitializeAnalyzer();
```

```
// Create a report object
CatalogItem report = new CatalogItem();
report.ItemType = CatalogItemType.Report;
report.Name = "Report with Cubes";
report.ParentId = analyzerObject.FindCatalogItemByName(secContext, "Shared Reports")[0].Id;

// Retrieve the DataSource Id
int dataSourceId = analyzerObject.FindCatalogItemByName(secContext, "Test Data Source")[0].Id;
// The test data source refers to the server "OLAPSVR1"
// This Analysis Server has the Adventure Works DW database.

// Call CreateReport() method to create an in-memory report instance
ReportInstance reportInstance = analyzerObject.CreateReport(secContext, report, false);

// Retrieve the name of the first Pivot Table in the report
string pivotTableName = reportInstance.ComponentNames[0];

// Initialize the cube with the details. This will update the report with the cube details.
reportInstance = analyzerObject.InitializeCube(secContext, reportInstance, pivotTableName,
                  dataSourceId, "Adventure Works DW", "Sales");

// Create an array of toolbar buttons having the buttons that need to be hidden
ToolbarButton[] hiddenButtons = new ToolbarButton[]
      {
            ToolbarButton.DataSource,
            ToolbarButton.SchemaTree,
            ToolbarButton.Save
```

```
        };

// Hide the toolbar buttons
reportInstance = analyzerObject.HideToolbarButtons(secContext, reportInstance, hiddenButtons);
```

4.14 Import functionality related APIs

4.14.1 XML Sample

Sample format of the exported XML file with reports which can be generated by using the export option from Analyzer user interface.

```
<Reports>
    <DataSource DataSrcId="-727046365" DataSrcName="DEFAULT" Description="Initial Default Databasource" SourceType="1"
        ServerName="OLAPSVR1" InitialCatalog="" CertType="2" UserId="" Password="PT1BR0FBQUFJQUFBQVl4aWY5Y=="
        FolderId="937150579" Inherited="Y" CreateDate="" OwnerId=" DOMAIN\USER" LastModifiedDate="" ModifiedId="0" />
    <Report ReportId="120390503" ReportName="Report for Import" Description="" FolderId="624687020" Inherited="Y"
        ModifiedId="0" OwnerId="DOMAIN\USER">
        <Definition>AAEAAAD/////AQAAAAAAAAAMAgAAAGNCdXNpbmVzc09ubGluZS5EYXNoYm9hcmQu==</Definition>
    </Report>
    <Bookmark BookmarkId="-1157245458" BookmarkName="Test Bookmark" Description="" ReportId="120390503" OwnerId="
    DOMAIN\USER " IsPublic="Y">
        <Definition>AAEAAAD/////AAAAAMAgAAAGNCdXNpbmVzc09ubGluZS5EYXNoYm9hcmQuU3RkSW==</Definition>
    </Bookmark>
</Reports>
```

4.14.2 ImportDataSource Method

This method is used to import a datasource from the XML file which has been exported using export report option from Analyzer. This option is available in Analyzer user interface and there is no API available for the export functionality. All the parameters that has to be passed on to this method can be retrieved from the exported XML file.

Syntax:

```
ImportDataSource(SecurityContext context, int id, string name, string description,
                 int dataSourceType, string serverName, string initialCatalog,
                 int certType, int ownerId, int parentId, bool inherited)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
id
```

The ID of the datasource that needs to be imported.

`name`

The name of the datasource that needs to be imported.

`description`

The description of the datasource that is being imported.

`dataSourceType`

The data source type (Analysis Server version) of the datasource that is being imported.

`serverName`

The name of the server corresponding to the datasource that is being imported.

`initialCatalog`

This field is not being used as of now as the exported XML contains an empty string for this value.

`certType`

The is the value of the attribute value "CertType" under the node "DataSource" the the exported XML file.

`ownerId`

The user Id in analyzer for the user who created the DataSource. If you wish to have a different owner Id, then you can specify a different user Id available in Analyzer application.

`parentId`

The folder Id under which the data source has to be imported. This value also can be retrieved from the XML file. Usually it's the Shared Data Sources folder in this context.

`inherited`

The flag indicating whether the object is inherited or not. This value will always be "Y" for the datasource and can be retrieved from the XML file.

Return Type

This method returns object of BaseState class which can be used to test whether the call was successful or not.

Examples:

The following code shows how to Import a data source from the XML file mentioned at the starting of this section.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Load the contents of the XML file
HtmlInputFile file = ""; // load the content of the XML file into this field
XmlTextReader reader = new XmlTextReader(file.PostedFile.InputStream);
XmlDocument ReportSource = new XmlDocument();
ReportSource.Load(reader);

int dataSourceId = "12365481";
// Get the node for the specific datasource that is being imported
string nodePath = string.Format("/Reports/DataSource[@DataSrcId={0}]", dataSourceId);
XmlNode dataSourceNode = ReportSource.SelectSingleNode(nodePath);

// Invoke the API for importing a datasource
analyzerObject.ImportDataSource(secContext,
                                Convert.ToInt32(dataSourceNode.Attributes["DataSrcId"].Value),
                                dataSourceNode.Attributes["DataSrcName"].Value,
                                dataSourceNode.Attributes["Description"].Value,
                                Convert.ToInt32(dataSourceNode.Attributes["SourceType"].Value),
                                dataSourceNode.Attributes["ServerName"].Value,
                                dataSourceNode.Attributes["InitialCatalog"].Value,
                                Convert.ToInt32(dataSourceNode.Attributes["CertType"].Value),
                                "DOMAIN/USER",
                                Convert.ToInt32(dataSourceNode.Attributes["FolderId"].Value),
                                dataSourceNode.Attributes["Inherited"].Value.Equals("Y"));
```

4.14.3 ImportFolder Method

This method is used to import a folder from the XML file which has been exported using export report option from Analyzer. This option is available in Analyzer user interface and there is no API available for the export functionality. All the parameters that has to be passed on to this method can be retrieved from the exported XML file.

Syntax:

```
ImportDataSource(SecurityContext context, int id, string name, string description,
                 int ownerId, int parentId, bool inherited)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
id
```

The ID of the folder that needs to be imported.

```
name
```

The name of the folder that needs to be imported.

```
description
```

The description of the folder that is being imported.

```
ownerId
```

The user Id in analyzer for the user who created the Folder. If you wish to have a different owner Id, then you can specify a different user Id available in Analyzer application.

```
parentId
```

The parent folder Id under which the folder has to be imported. This value also can be retrieved from the XML file.

```
inherited
```

The flag indicating whether the object is inherited or not. This value can be retrieved from the XML file.

Return Type

This method returns object of BaseState class which can be used to test whether the call was successful or not.

Examples:

The following code shows how to Import a Folder from the XML file mentioned at the starting of this section.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
// Load the contents of the XML file
HtmlInputFile file = ""; // load the content of the XML file into this field
XmlTextReader reader = new XmlTextReader(file.PostedFile.InputStream);
XmlDocument ReportSource = new XmlDocument();
ReportSource.Load(reader);

int folderId = 23654782;
// Get the node for the specific Folder that is being imported
string nodePath = string.Format("/Reports/Folder[@FolderId={0}]", folderId);
XmlNode folderNode = ReportSource.SelectSingleNode(nodePath);

int ownerId = analyzerObject.FindUserByName(secContext, folderNode.Attributes["OwnderId"].Value).Id;

// Invoke the API for importing a Folder
analyzerObject.ImportFolder(secContext,
                            Convert.ToInt32(folderNode.Attributes["FolderId"].Value),
                            folderNode.Attributes["FolderName"].Value,
                            folderNode.Attributes["Description"].Value,
                            ownerId,
                            Convert.ToInt32(folderNode.Attributes["ParentFolderId"].Value),
                            // you can specify a different parent folder Id if you want
                            folderNode.Attributes["Inherited"].Value.Equals("Y"));
```

4.14.4 ImportReport Method

This method is used to import a report from the XML file which has been exported using export report option from Analyzer. This option is available in Analyzer user interface and there is no API available for the export functionality. All the parameters that has to be passed on to this method can be retrieved from the exported XML file.

Syntax:

```
ImportReport(SecurityContext context, int id, string name, string description, int ownerId, int parentId,
             bool inherited, int handle)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
id
```

The ID of the report that needs to be imported.

```
name
```

The name of the report that needs to be imported.

```
description
```

The description of the report that is being imported.

```
ownerId
```

The user Id in analyzer for the user who created the report. If you wish to have a different owner Id, then you can specify a different user Id available in Analyzer application.

```
parentId
```

The folder Id under which the report has to be imported. This value also can be retrieved from the XML file.

```
inherited
```

The flag indicating whether the object is inherited or not. The value for this can be retrieved from the XML file.

```
handle
```

This is an integer number acts as a handler for importing the definition of the report. This value can be retrieved using the methods CreateDefinition64 and AppendDefinition64 which is explained in the further sections.

Return Type

This method returns object of BaseState class which can be used to test whether the call was successful or not.

Examples:

The following code shows how to Import a report from the XML file mentioned at the starting of this section.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Load the contents of the XML file
HtmlInputFile file = ""; // load the content of the XML file into this field
XmlTextReader reader = new XmlTextReader(file.PostedFile.InputStream);
XmlDocument ReportSource = new XmlDocument();
ReportSource.Load(reader);

int reportId = 23654785;
// Get the node for the specific report that is being imported
string nodePath = string.Format("/Reports/Report[@ReportId={0}]", reportId);
XmlNode reportNode = ReportSource.SelectSingleNode(nodePath);

int ownerId = analyzerObject.FindUserByName(secContext, reportNode.Attributes["OwnderId"].Value).Id;

// Get a handle for importing the definition
string reportDefinition = reportNode.FirstChild.InnerText;
int tempLength = 10000;
int count = reportDefinition.Length / tempLength + 1;
string subDefinition = string.Empty;
int handle = analyzerObject.CreateDefinition64();
for (int i = 0; i < count; i++)
      {
```

```
            if ((i + 1) * tempLength > reportDefinition.Length)
                  subDefinition = reportDefinition.Substring(i * tempLength);
            else
                  subDefinition = reportDefinition.Substring(i * tempLength, tempLength);
            analyzerObject.AppendDefinition64(handle, subDefinition);
      }

// Invoke the API for importing a report
analyzerObject.ImportReport(secContext,
                              Convert.ToInt32(reportNode.Attributes["ReportId"].Value),
                              reportNode.Attributes["ReportName"].Value,
                              reportNode.Attributes["Description"].Value,
                              ownerId,
                              Convert.ToInt32(reportNode.Attributes["FolderId"].Value),
                              // you can specify a different parent folder Id if you want
                              reportNode.Attributes["Inherited"].Value.Equals("Y"),
                              handle);
```

4.14.5 ImportReportWithDataSourceInfo Method

This method is used to import a report from the XML file which has been exported using export report option from Analyzer with the option of specifying a different DataSource or Database or Cube. For example, if you want to import a report into a different version Analyzer and you have datasource/database/cube different from the one where the report has been exported from, then you can specify the appropriate datasource/database/cube so that the import report will be associated with these details.

Syntax:

```
ImportReport(SecurityContext context, int id, string name, string description, int ownerId, int parentId,
             bool inherited, int handle , int dataSourceId, string asDatabaseName, string asCubeName)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
id
```

The ID of the report that needs to be imported.

```
name
```

The name of the report that needs to be imported.

```
description
```

The description of the report that is being imported.

```
ownerId
```

The user Id in analyzer for the user who created the report. If you wish to have a different owner Id, then you can specify a different user Id available in Analyzer application.

```
parentId
```

The folder Id under which the report has to be imported. This value also can be retrieved from the XML file.

```
inherited
```

The flag indicating whether the object is inherited or not. The value for this can be retrieved from the XML file.

```
handle
```

This is an integer number acts as a handler for importing the definition of the report. This value can be retrieved using the methods CreateDefinition64 and AppendDefinition64 which is explained in the further sections.

```
dataSourceId
```

The Id of the dataSource which should be associated with the report that is being import if the datasource is different. The you want to use the same datasource Id of the report then pass the parameter as -1.

```
inherited
```

Name of the olap database that the report should be associated with. If there is no change in the database name where the report is being imported then pass blank string as parameter.

```
inherited
```

Name of the olap cube that the report should be associated with. If there is no change in the cube name where the report is being imported then pass blank string as parameter.

Return Type

This method returns object of BaseState class which can be used to test whether the call was successful or not.

Examples:

The following code shows how to Import a report from the XML file mentioned at the starting of this section. Here the datasource id is left unchanged and the database/cube names are being changed.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Load the contents of the XML file
HtmlInputFile file = ""; // load the content of the XML file into this field
XmlTextReader reader = new XmlTextReader(file.PostedFile.InputStream);
XmlDocument ReportSource = new XmlDocument();
ReportSource.Load(reader);

int reportId = 23654785;
// Get the node for the specific report that is being imported
string nodePath = string.Format("/Reports/Report[@ReportId={0}]", reportId);
XmlNode reportNode = ReportSource.SelectSingleNode(nodePath);

int ownerId = analyzerObject.FindUserByName(secContext, reportNode.Attributes["OwnderId"].Value).Id;

// Get a handle for importing the definition
string reportDefinition = reportNode.FirstChild.InnerText;
int tempLength = 10000;
int count = reportDefinition.Length / tempLength + 1;
string subDefinition = string.Empty;
int handle = analyzerObject.CreateDefinition64();
for (int i = 0; i < count; i++)
```

```
        {
                if ((i + 1) * tempLength > reportDefinition.Length)
                        subDefinition = reportDefinition.Substring(i * tempLength);
                else
                        subDefinition = reportDefinition.Substring(i * tempLength, tempLength);
                analyzerObject.AppendDefinition64(handle, subDefinition);
        }

// Invoke the API for importing a report with data source info
analyzerObject.ImportReportWithDataSourceInfo(secContext,
                                Convert.ToInt32(reportNode.Attributes["ReportId"].Value),
                                reportNode.Attributes["ReportName"].Value,
                                reportNode.Attributes["Description"].Value,
                                ownerId,
                                Convert.ToInt32(reportNode.Attributes["FolderId"].Value),
                                // you can specify a different parent folder Id if you want
                                reportNode.Attributes["Inherited"].Value.Equals("Y"),
                                handle,
                                -1, // leaving the data source id unchanged
                                "New Database",
                                "New Cube"
                                );
```

4.14.6 ImportBookmark Method

This method is used to import a bookmark for a specific report from the XML file which has been exported using export report option from Analyzer.

Syntax:

```
ImportReport(SecurityContext context, int id, string name, string description, int ownerId, int reportId,
             bool isPublic, int handle)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
id
```

The ID of the report that needs to be imported.

```
name
```

The name of the report that needs to be imported.

```
description
```

The description of the report that is being imported.

```
ownerId
```

The user Id in analyzer for the user who created the report. If you wish to have a different owner Id, then you can specify a different user Id available in Analyzer application.

```
reportId
```

The report Id under which is associated with the bookmark that is being imported. This value also can be retrieved from the XML file.

```
isPublic
```

The flag indicating whether the bookmark is public or not. The value for this can be retrieved from the XML file.

```
handle
```

This is an integer number acts as a handler for importing the definition of the bookmark. This value can be retrieved using the methods CreateDefinition64 and AppendDefinition64 which is explained in the further sections.

Return Type

This method returns object of BaseState class which can be used to test whether the call was successful or not.

Examples:

The following code shows how to Import a report from the XML file mentioned at the starting of this section.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();

// Load the contents of the XML file
HtmlInputFile file = ""; // load the content of the XML file into this field
XmlTextReader reader = new XmlTextReader(file.PostedFile.InputStream);
XmlDocument ReportSource = new XmlDocument();
ReportSource.Load(reader);

int reportId = 23654785;
// Get the node for the specific bookmark based on the report Id
string nodePath = string.Format("/Reports/Bookmark[@ReportId={0}]", reportId);
XmlNode bookmarkNode = ReportSource.SelectSingleNode(nodePath);

int ownerId = analyzerObject.FindUserByName(secContext, bookmarkNode.Attributes["OwnderId"].Value).Id;

// Get a handle for importing the definition
string bookmarkDefinition = bookmarkNode.FirstChild.InnerText;
int tempLength = 10000;
int count = bookmarkDefinition.Length / tempLength + 1;
string subDefinition = string.Empty;
int handle = analyzerObject.CreateDefinition64();
for (int i = 0; i < count; i++)
      {
            if ((i + 1) * tempLength > bookmarkDefinition.Length)
                  subDefinition = bookmarkDefinition.Substring(i * tempLength);
            else
                  subDefinition = bookmarkDefinition.Substring(i * tempLength, tempLength);
```

```
            analyzerObject.AppendDefinition64(handle, subDefinition);
        }

    // Invoke the API for importing a datasource
    analyzerObject.ImportBookmark(secContext,
                                  Convert.ToInt32(bookmarkNode.Attributes["ReportId"].Value),
                                  bookmarkNode.Attributes["BookmarkName"].Value,
                                  bookmarkNode.Attributes["Description"].Value,
                                  ownerId,
                                  Convert.ToInt32(bookmarkNode.Attributes["ReportId"].Value),
                                  bookmarkNode.Attributes["IsPublic"].Value.Equals("Y"),
                                  handle);
```

4.14.7 ListReportNullRefDataSources Method

This method is used to get a list of datasources that are associated with a report but not available in the Analyzer application instance.

Syntax:

```
ImportReport(SecurityContext context, int reportId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

```
reportId
```

The report Id for which the datasource null reference needs to be checked.

Return Type

This method returns object of CatalogItem which will have the datasource objects.

Examples:

The following code shows how to retrieve the list of datasources associated with the report "23654785" but are not present in the Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
CatalogItem[] missingDataSources = analyzerObject.ListReportNullRefDataSources(secContext, 23654785);
```

4.14.8 CreateDefinition64 Method

This method is used to get a handler which can be used for importing definitions like report definition or bookmark definition.

Syntax:

```
CreateDefinition64()
```

Arguments:

This method does not need any input parameter.

Return Type

This method returns a handler of type integer.

Examples:

The usage of this method can be found in the example of APIs InportReport, ImportReportWithDataSourceInfo and ImportBookmark.

4.14.9 AppendDefinition64 Method

This method is used to associate the handler with the definition of a report/bookmark.

Syntax:

```
AppendDefinition64(int handle, string definition64)
```

Arguments:

`context`

The security context under which this action needs to be performed.

`definition64`

Specific part of the report/bookmark definition which will be associated with the handler.

Return Type

This method returns a BaseState object which can be used to test if the call was successful or not.

Examples:

The usage of this method can be found in the example of APIs InportReport, ImportReportWithDataSourceInfo and ImportBookmark.

4.15 Subscription Management APIs

4.15.1 GetSubscriptions Method

This method retrieves a collection of subscriptions available in the Analyzer application instance for the specific security context.

Syntax:

```
GetSubscriptions(SecurityContext context)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

Return Type

This method returns an array of Subscription objects.

Examples:

The following code gets the collection of Subscriptions available in Analyzer application instance.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
Subscription[] subscriptions = analyzerObject.GetSubscriptions(secContext);
```

4.15.2 TriggerSubscription Method

This method explicitly executes/runs a specific subscription based on the subscription Id passed in as parameter.

Syntax:

```
TriggerSubscription(SecurityContext context, int subscriptionId)
```

Arguments:

```
context
```

The security context under which this action needs to be performed.

`subscriptionId`

The Id of the subscription which needs to be executed explicitly.

Return Type

This method returns a BaseState object.

Examples:

The following code explicitly executes the subscription "123456" and returns the result in the BaseState object.

```
Analyzer2005 analyzerObject = InitializeAnalyzer();
BaseState baseState = analyzerObject.TriggerSubscription(secContext, 123456);
```