Business Objects™

crystalreports.com Web Services Partner API

Insert CRDC Functionality in Your Application

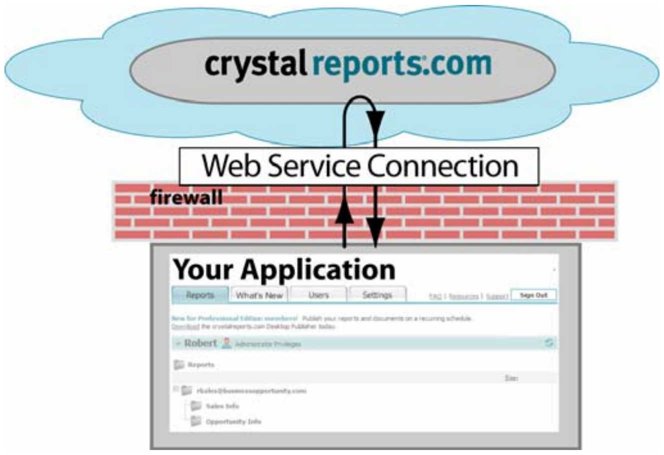
Version 1.

May 2007

Table of Contents

Introduction

The Business Objects Partner API allows partner's developers to insert crystalreports.com (CRDC) functions into their applications. Using a set of web service calls to CRDC developers can upload new files, download existing files, or view documents in CRDC. These calls mean you can distribute files and reports which provide intelligence for sales quoting, sales tracking, support tracking and many other applications.



CRDC Functionality Supported by the Partner API

There are a number of CRDC functions you can use within your own application:

- Upload files to create a new object (or a new historical instance of an object)
- Upload a new file to overwrite the existing file of an object
- Download files from an existing object
- Get the viewer URL for any viewable object. This URL can then be used in the browser to view the document in CRDC.

NOTE: Currently only .rpt files can be uploaded. All supported CRDC files can be downloaded using the API.

Web Services Overview

Web Services are a standard way for providing application programming interfaces (API) over the Internet. The Partner API described in this document follows the industry standard model. The CRService Web Service Definition Language (WSDL) module provides a set of Application Programmer Interfaces and structures which developers use to communicate requests and information between CRDC and their application.

For more information about WSDL, see http://www.w3.org/TR/wsdl

This document describes the operations a partner application may request from CRDC.

Using Business Objects Web Services

To access the Business Objects Partner WS API, you will need to obtain the Partner Web Service WSDL file. The WSDL file defines the Web Service requests that CRDC supports. Developers should note that they must use a secure transport layer (web services using HTTPS as CRDC only accepts HTTPS connections.

Overview for Using Business Objects Web Services

The process for adding Business Objects Partner web services to an application includes the following steps. It is easiest to use a web services library such as Axis2 (for Java) or (WSE 3.0) for .NET 2.0. These libraries provide a WSDL utility for generating the client stubs in your development environment.

1. *Download the Business Objects Partner WS API WSDL package (https://na.crystalreports.com/wsdl/CRService.wsdl)*
2. *Generate client stubs from the WSDL in your development environment*
3. *Use CRDC functions in your application*
4. *Test*
5. *Deploy*

CRDC and the Partner API use cookies to track the client's session, developers and users will need to make sure that their web service client code is returning cookies back to CRDC or they will only be able to login to the server.

The Crystal Reports Service WSDL

The CRService WSDL describes the objects and methods for interacting with the Web service APIs. The attributes of the objects are exposed to communicate between CRDC and your application.

Overview of CRDC and Your Consumer Application's Interactions

The interactions between your application and CRDC are fairly simple. This overview describes the flow of interactions.

Login

On Login your application will receive a CRLoginInfo object from the server. This object contains information about the user's CRDC account.

NOTE: Future releases may also give the user access to accounts to which he or she has been invited.

Navigating Folders in CRDC

Included in the CRLoginInfo object's information is the RootFolderId from the accessible account. The root folder id may be used in a call to GetObjects( ) to retrieve the root folder from the server as a CRObject.

Each CRObject has a Children property which contains an array of object ids. This array gives you the Ids of all the children which belong to the object. Use these arrays of ids in calls to GetObjects( ) to retrieve these child CRObjects. If the array is empty then the object has no children. Using this means of retrieving child objects you can navigate down the folder tree. CRObjects also has a ParentId property which can be used for navigating up the folder tree.

You will notice if you look at the sample application, the use of a dummy node. This dummy node allows the viewer to display a "+" symbol next to the folder, so the objects may be retrieved upon opening of the folder, rather than attempting to retrieve all items of the tree upon first login.

Viewing Documents in CRDC

To view a document from CRDC, you call GetViewerUrl() with the id of the object. The application is returned a url into the CRDC website which can be put into the browser. If that browser window has not been logged in from, then the user will be prompted to log in first before being sent to the viewer to see the object. If the browser window was logged in from, then the user is sent directly to the viewer.

NOTE: Currently there are no options for creating your own custom "skins" for the viewer. Custom skins are possible, but require additional custom development. If you are interested in learning about skinning or user interface customization contact your Business Objects Partner Manager.

Mechanism for Uploading, Downloading and Replacing Files

The routines for uploading, downloading, or replacing files use the SOAP Message Transmission Optimization Mechanism (MTOM).

Uploading Files

To add a new document to CRDC the application sets up a CRNewObjectOptions object which is then used in a call to CreateObjectMtom(). The CRNewObjectOptions specifies the new object's name, the id of the folder to which the object will be added, and the CRFileInfo object containing the file name and the file data the application is uploading. The CreateObjectMtom( ) routine returns a CRObject for the newly added object.

NOTE: Your application may take advantage of Business Objects Open Data Connector for .rpt files.

To add the new object as an historical instance of an existing object rather than adding it as a new top level object in a folder, set the ParentId to the Id of an existing object. Historical instances allow you to keep multiple versions of the same report, each containing data from different dates.

NOTE: With this initial version of the Partner API you can only add instances of Crystal Reports documents (.rpt files) documents, and not Microsoft Word, Excel or other document types.

Downloading Files

To download a file, call DownloadFileMtom() with the object's Id. A CRFileInfo object is returned. The CRFileInfo object contains the file name, size, and the file itself (in binary format).

Replacing Files

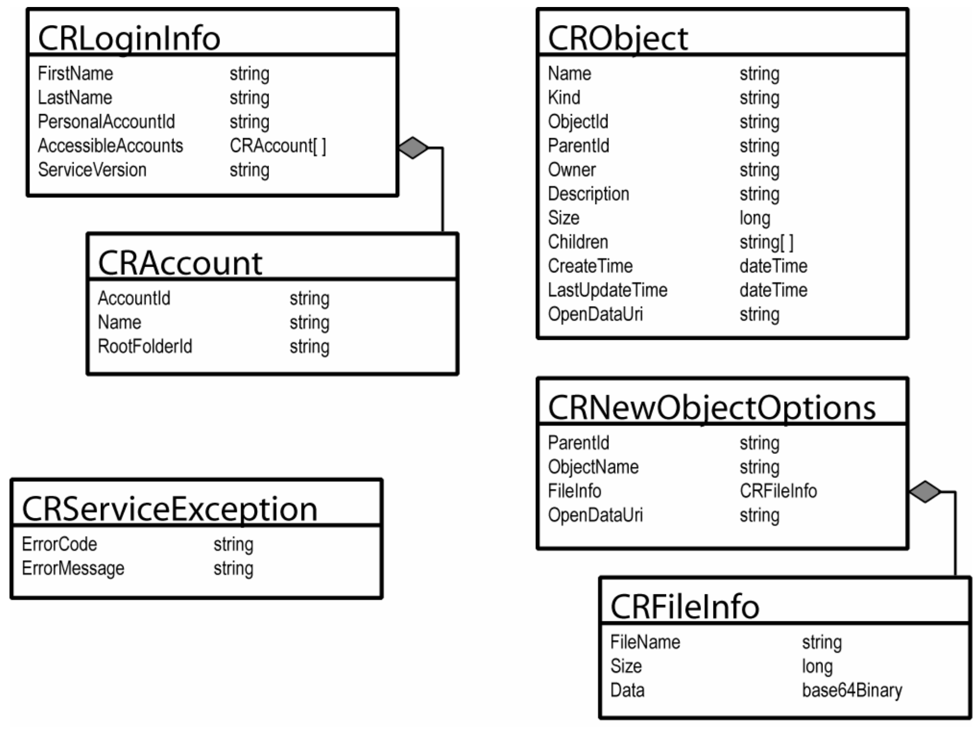
To replace the file of an existing object, call ReplaceFileMtom() with the Id of an existing object and a CRFileInfo object containing the new file name and data. This allows you to keep an object in CRDC, but update it with new data.

Error Reporting

Each method of the CRService WS API will return a CRServiceException fault if it encounters any problems in the course of it's execution. The CRServiceException contains an error code string to programmatically identify the type of error that occurred. It also includes the corresponding error message.

Objects

The object model for the Partner WS API is quite straight forward. CRLoginInfo contains administration information for the user, including accounts to which the user has access. CRObject is a tree of objects. The Children attribute allows you to consume objects down the tree. ParentId allows for navigating back up the tree. CRNewObjectOptions is used for uploading new objects.



CRLoginInfo

CRLoginInfo is used to describe the user, the account, possible other accounts which the user may have access to, and the version of the service which your customer application is using. As new versions of the service are released you may upgrade to the new features and functionality. However, older versions will continue to be supported.

Attribute	Type	Description
FirstName	string	The first name of the user
LastName	string	The last name of the user
PersonalAccountID	string	The account ID of the user. This is the account of the user who is logged into your application and sent to CRDC.
AccessibleAccounts	CRAccount[ ]	The accounts which the user may access. In this first version of the Partner API users only have access to their own account. In future versions of the Partner API, users will also be able to have access to accounts to which they are invited.

Attribute	Type	Description
ServiceVersion	string	The version of the service you are using.

CRAccount

CRAccount is used to track the accounts which the user has access to along with the root folder for each account. The root folder is the initial starting place for navigating the folder structure within CRDC.

Attribute	Type	Description
AccountId	string	The account ID for this account
Name	string	The name of the account
RootFolderId	string	The root folder ID. The root folder is the default folder for the account – all objects are stored inside this folder.

CRObject

CRObject contains the ObjectId for displaying objects as well as the tree which represents the folder structure within CRDC. The Children attribute contains all child objects of the current object (when the length of Children is greater than 0 the object has at least one child). ParentId provides a mechanism for traversing up the tree.

Attribute	Type	Description
Name	string	The name of the object
Kind	string	The type of the object – Folder, CrystalReport, Word, etc.
ObjectID	string	The ID of the object. This ID is used for displaying objects in the viewer with the GetViewerUrl( ) method.
ParentID	string	The ID of the object which is the parent of the current object
Owner	string	The owner of the object
Description	string	The description of the object. For Crystal Report objects, this is the description value in the .rpt file.
Size	long	The size (in bytes) of the object's file.
Children	string[ ]	An array of IDs of child objects of this object
OpenDataUri	string	The Open Data Connector url that this object is using. Currently this is only applicable to Crystal Report objects.

Attribute	Type	Description
CreateTime	dateTime	Time when the object was first uploaded
LastUpdateTime	dateTime	Time when the object was last modified

CRNewObjectOptions

CRNewObjectOptions is used to specify properties of new objects.

Attribute	Type	Description
ParentId	string	The new object will be created as a child of the object with this Id.
ObjectName	string	The name of the new object.
OpenDataUri	string	Optional. The url for connecting to your Open Data Connector provider. Currently this is only applicable to Crystal Report objects.
FileInfo	CRFileInfo	The information about the file being uploaded.

CRFileInfo

CRFileInfo contains information about a file being uploaded or downloaded.

Attribute	Type	Description
Filename	string	The name of the file.
Size	long	The size of the file (in bytes).
Data	Base64Binary	The actual data.

CRServiceException

Attribute	Type	Description
ErrorCode	string	The unique code that identifies what type of error has occurred.
ErrorMessage	string	The error message.

Method Signatures

Login

Parameter	Type	Description
Username	string	The login id of the user.
Password	string	The password.
Locale	string	Locale is not used in this version.

Logout

Log out from crystalreports.com web service.

GetObjects

Retrieve a set of objects. The GetObjects() method allows up to 100 objects at a time. If more than 100 IDs are passed, then an exception will be thrown.

Attribute	Type	Description
ObjectID	String[ ]	IDs of objects to get

CreateObjectMtom

Upload a file using SOAP Message Transmission Optimization Mechanism (MTOM)

Attribute	Type	Description
CRNewObjectOptions	CRNewObjectOptions	The collection of values to use when creating the new object.

ReplaceFileMtom

Upload a new file using MTOM to replace an existing object's file.

Attribute	Type	Description
ObjectId	String	The Id of the object whose file will be overwritten.
FileInfo	CRFileInfo	The file to upload.

DownloadFileMtom

Download the file of an object using MTOM.

Attribute	Type	Description
ObjectID	string	The Id of the object whose file will be downloaded.

GetViewerUrl

Get the url of a viewable object that can be used in a web browser to view the object in CRDC.

Attribute	Type	Description
ObjectID	string	The Id of the object to view.

Sample Application

In the Partner API package is a sample application. This sample application is just one way of exposing the Partner Web Services API to users. Because you are building around a WSDL, your application may be a Web application, a .Net application, a Java application, or other type of application.

The sample application is a .NET 2.0 windows application that is composed of 2 main parts. The first part is the CRServiceConsumer project which contains the WSE 3.0 generated client side code along with a thin wrapper class. This project gets built into the CRServiceConsumer.dll which is used by the second part of the sample application, the CRServiceTest project. The CRServiceTest project is the actual windows application that implements all user interface and the logic of interacting with CRDC.

As mentioned earlier in this document, a web service library should be used to generate client side code from the published WSDL file. This sample application uses the WSE 3.0 library for .NET 2.0 (http://www.microsoft.com/downloads/details.aspx?familyid=018a09fd-3a74-43c5-8ec1-8d789091255d&displaylang=en). Once WSE 3 has been downloaded and installed on your Visual Studio 2005 machine, you can create a web reference in your project and point it to the CRService WSDL. This will automatically generate all the client side code for you. The one that you will need to worry about is the CRServiceWse class that inherits from Microsoft.Web.Services3.WebServicesClientProtocol.

```
public partial class CRServiceWse :
Microsoft.Web.Services3.WebServicesClientProtocol
```

You want to make sure that you use this class as it is the one generated by WSE 3. If you look closely, you will also notice there is a CRService class that inherits from System.Web.Services.Protocols.SoapHttpClientProtocol which is the class generated by the default web services library that comes with Visual Studio 2005. It is important to use the CRServiceWse class as it supports the MTOM binary transfer mechanism while the CRService class does not.

Once you have the CRServiceWse class generated, you can start using it directly in your application. Instead of using it directly, this sample application chose to write a thin wrapper class around the CRServiceWse class. The wrapper class is used by the application logic and hides some of the web services implementation and error handling. The wrapper class is called CRServiceConsumer and is described in more detail below.

CRServiceConsumer

Constructor

Below is the code for the CRServiceConsumer constructor. It is responsible for instantiating and configuring the CRServiceWse object.

- The url parameter is the url to the service which can be found in the service's port address attribute of the WSDL file.
- The cookie container is used to handle the cookies returned by the login method. This cookies must be returned to the server on each subsequent web service call so that your session can be retrieved.
- The RequireMtom property is set to true to indicate that you want to use MTOM binary data transfer.

```
public CRServiceConsumer(string url)
{
	m_impl = new CRServiceWse();
	m_impl.Url = url;
	m_impl.Credentials = System.Net.CredentialCache.DefaultCredentials;
	m_impl.CookieContainer = new CookieContainer();
	m_impl.RequireMtom = true;
	…..
}
```

Login method

The Login method is quite simple; it just passes the users credentials straight into the web service call.

- As mentioned earlier in this document, the locale parameter is not actually used at this time so you can simply pass in an empty string.
- The CRServiceException class encapsulates the error code and message that are returned from CRDC if an error occurs.

```
public CRLoginInfo Login(string userName, string password, string locale)
{
	try
	{
		CRLoginInfo loginInfo = m_impl.Login(userName, password, locale);
		return loginInfo;
	}
	catch (SoapException ex)
	{
		throw new CRServiceException(ex.Message, ex);
	}
}
```

GetObjects method

The GetObjects method is also very simple.

```
public CRObject[] GetObjects(string[] objectIds)
```

```
{
        try
        {
                CRObject[] objs = m_impl.GetObjects(objectIds);
                return objs;
        }
        catch (SoapException ex)
        {
                throw new CRServiceException(ex.Message, ex);
        }

}
```

CreateObjectMtom method

This method is a bit more complex as it is responsible for building up the CRNewObjectOptions object which gets passed into the web service call.

- The OpenDataUri property is optional so you can ignore it altogether if you don't use it.

```
public CRObject CreateObjectMtom(string parentId,
                                  string objectName,
                                  FileInfo file,
                                  string openDataUri)
{
        try
        {
                // Create a CRFileInfo
                CRFileInfo fileInfo = createCRFileInfo(file);

                // Create the options
                CRNewObjectOptions opts = new CRNewObjectOptions();
                opts.ParentId = parentId;
                opts.ObjectName = objectName;
                opts.FileInfo = fileInfo;
                opts.OpenDataUri = openDataUri;

                // Do the upload
                CRObject obj = m_impl.CreateObjectMtom(opts);
                return obj;
        }
        catch (SoapException ex)
        {
                throw new CRServiceException(ex.Message, ex);
        }

}
```

ReplaceFileMtom method

This method will create the CRFileInfo object using a private helper method and use it in the web service call.

```
public void ReplaceFileMtom(string objectId,
                            FileInfo file)
{
        try
        {
                CRFileInfo fileInfo = createCRFileInfo(file);

                // Do the upload
```

```
            m_impl.ReplaceFileMtom(objectId, fileInfo);
        }
        catch (SoapException ex)
        {
            throw new CRServiceException(ex.Message, ex);
        }

}
private CRFileInfo createCRFileInfo(FileInfo localFile)
{
        FileStream inStream = null;
        try
        {
            // Read in the data.
            inStream = localFile.OpenRead();
            byte[] data = new byte[localFile.Length];
            int bytesRead = inStream.Read(data, 0, data.Length);

            // Create a CRFileInfo
            CRFileInfo fileInfo = new CRFileInfo();
            fileInfo.FileName = localFile.Name;
            fileInfo.Data = data;

            return fileInfo;
        }
        finally
        {
            if (inStream != null)
                inStream.Close();
        }

}
```

DownloadFileMtom method

The wrapper class doesn't do too much work for this method.

```
public CRFileInfo DownloadFileMtom(string objectID)
{
        try
        {
            // Do the download
            CRFileInfo fileInfo = m_impl.DownloadFileMtom(objectID);
            return fileInfo;
        }
        catch (SoapException ex)
        {
            throw new CRServiceException(ex.Message, ex);
        }

}
```

CRServiceTest

Login and get the account's root folder

This method will take the user's credentials from a pair of text boxes in the UI and login to CRDC. It will save the account's root folder id into a text box which the user can edit if they choose.

```
private void doLogin()
{
	…
	m_consumer = new CRServiceConsumer(url);
	CRLoginInfo loginInfo = m_consumer.Login(m_userNameTextBox.Text,
	                                   m_passwordTextBox.Text, LOCALE);
	…
	m_objectIDTextBox.Text = loginInfo.AccessibleAccounts[0].RootFolderId;
	…
}
```

Retrieve an arbitrary object

This method will take an arbitrary object id from a text box in the UI and will call the GetObjects method. The createTreeNode method will return a tree node with the appropriate name that can be inserted into the tree. If the CRObject has any children then the tree node will have a dummy node attached as a child so that the little "+" icon is shown in the UI. This saves having to get all the child objects to fully build the tree right away. Instead the child objects can be retrieved if and when the current node is actually expanded (see the next section below).

```
private void m_getObjectButton_Click(object sender, EventArgs e)
{
	string objectID = m_objectIDTextBox.Text;
	…
	try
	{
		Cursor = Cursors.WaitCursor;
		CRObject[] objs = m_consumer.GetObjects(new string[]{objectID});
		…

		// Add a node to the tree in the UI.
		m_objTree.Enabled = true;
		m_objTree.BeginUpdate();
		m_objTree.Nodes.Clear();
		m_objTree.Nodes.Add(createTreeNode(objs[0]));
		m_objTree.ImageList = OBJECT_IMAGE_LIST;
		m_objTree.EndUpdate();
	}
	catch (Exception ex)
	{
		…
	}
	finally
	{
		Cursor = Cursors.Default;
	}
}
```

Get an object’s child objects

This method gets called whenever a node in the UI tree is expanded. It checks the selected tree node’s child nodes to see if they are real or not. If they are real then we don’t need to do anything as they have already been retrieved from CRDC. If the child nodes collection only contains the dummy node then we need to call GetObjects to get the real children from CRDC.

```
private void m_objTree_BeforeExpand(object sender, TreeViewCancelEventArgs e)
{
      // See if the node that is expanding has real child nodes
      // or our temporary dummy node.
      TreeNode expandingNode = e.Node;
      TreeNodeCollection nodes = expandingNode.Nodes;
      if (!(nodes.Count == 1 && nodes[0].Text == DUMMY_NODE_NAME))
      {
            // The children should be real so we don't need to do anything.
            return;
      }

      // We need to replace this dummy node with the real children.
      try
      {
            Cursor = Cursors.WaitCursor;
            m_objTree.BeginUpdate();
            nodes.Clear();

            CRObject[] crObjs =
m_consumer.GetObjects(((CRObject)expandingNode.Tag).Children);

            // Create nodes for the children.
            foreach (CRObject childObj in crObjs)
            {
                  TreeNode childNode = createTreeNode(childObj);
                  nodes.Add(childNode);
            }
      }
      catch (Exception ex)
      {
            …
      }
      finally
      {
            m_objTree.EndUpdate();
            Cursor = Cursors.Default;
      }
}
```

Index