Responsys Interact

API Guide — Standard

Version 6.19

Address permission requests, comments, or suggestions about Responsys Interact documentation to **docs@responsys.com**.

Responsys, Inc.
1100 Grundy Lane, 3rd Floor
San Bruno, CA 94066
(650) 745-1700

www.responsys.com

Contents

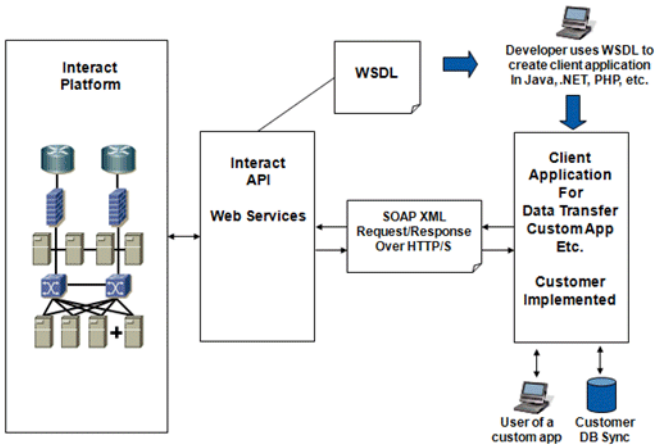
Introducing the Responsys Interact API

The Responsys Interact® Web Services Application Programming Interface (Interact API) gives you standards-based access to the data, content, and campaign management features of Responsys Interact. Using the Interact API, you can build solutions for marketing data automation, customize your campaign and content management processes, and remotely trigger events for recipients thereby entering them into Interact-based life cycle messaging programs.

Specifically, you may want to use the Interact API to:

- Synchronize marketing data between enterprise and partner systems
- Trigger individual email or mobile messages in response to some external event or activity detected by your web site or enterprise information systems
- Automate the import of creative content needed for your campaigns

This conceptual diagram shows how to use the Interact API.



Because the Interact API is based on a service-oriented architecture (SOA) and other industry-standard technologies such as SOAP and WSDL, your developers can use their choice of programming language and development environment to gain full programmatic access to your organization's Responsys Interact account. The Interact API supports easy integration of your enterprise systems with the campaigns and data stored in your Responsys Interact account - enabling greater automation of marketing tasks and processes.

Interact API functionality

The Interact API supports the following subset of the functionality of the Interact user interface and platform.

Session Management

- Login/Logout of an Interact API session
- Retrieving the current Interact platform timestamp

List and Data Management

- Insert, update, and delete records in Lists and Supplemental Tables

- Retrieve records from Lists, Supplemental Tables, and Profile Extension Tables
- Retrieve updated list member records

Content Management

- Create or delete document objects
- Set or get image files for a document object
- Set or get the markup content for a document object

Campaign Management

- Launch a campaign
- Get campaign launch status

Lifecycle Messaging Programs

- Trigger campaign messages to individual recipients
- Trigger custom events for individual recipients

About Interact API URLs

When your account is enabled for access to the Interact API, the Responsys Support team gives you the Web Services URLs you need to develop your projects. Depending on where your account is set up in the Responsys data center, you'll get Web Services URLs for the Interact 2 pod or the Interact 5 pod.

Development environments

The Interact API works with modern SOAP development environments, FOR EXAMPLE, Visual Studio .NET, Apache Axis, and others. Development platforms vary in their SOAP implementations and differences in implementation might prevent access to some or all of the features in the API. If you are using Visual Studio for .NET development, we recommend that you use Visual Studio 2003 or later.

Interact Platform maintenance and downtime

The Responsys Interact platform undergoes maintenance downtimes on a monthly or bi-monthly schedule. During these downtimes, Interact Campaign login sessions are not available. Attempts to create a login session during downtimes returns an error and client applications need to take the appropriate action, which may include alerts to support staff, integration job queuing, and/or scheduled re-tries.

Monitoring and throttling the frequency of API requests

Responsys monitors and throttles the frequency of API requests that are submitted from each Interact account. This is to ensure that the best possible level of service is offered to API clients in a shared environment. Depending on the type of API function, a specific frequency rate limit is imposed on the basis of an account's number of requests made per minute for that function. For example, the API function for triggering email messages can be called more times per minute than the API function for launching a campaign.

You choose from multiple tiers that accommodate standard, medium, or high volume API usage. When account exceeds its allowable frequency rate limit for an API request, you see the error code **API_LIMIT_EXCEEDED** and this message "**You exceeded your allowable limit to call the <function_name> API function. Please try again in a minute.**" If a specific user of an account is blocked from using selected API functions, the user sees the error code **API_BLOCKED** with this message: "**The <function_name> is currently not available to this user. Please contact tech support.**" See *Sample Code for Handling Exceeded Account Limits on page 57* for the appropriate block of sample code.

Backward compatibility

Responsys supports backward compatibility as new versions of the Interact API are released. This means that an application created to work with a given Interact API version will continue to work with *that same Interact API version* in future platform releases. Each version of the Interact API has a unique endpoint URL. Your applications will continue to work with the Interact API endpoint URLs of previous releases. You can migrate your client applications to newer Interact API version endpoint URLs to take advantage of enhanced functionality and bug fixes on a schedule that meets your needs.

Responsys does not guarantee that an application written against one Interact API version will work with future API versions, because changes in method signatures and data representations are often required to enhance the Interact platform. However, we strive to keep the Interact API consistent from version to version with minimal if any changes required to port applications to newer Interact API versions. When an API version is to be deprecated, advance end-of-life notice will be given at least 9 months before support for the API version is ended. Responsys will directly notify customers using API versions planned for deprecation.

Web service standards compliance

The Interact API was implemented in compliance with these specifications:

- Simple Object Access Protocol (SOAP) 1.1
  http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
- Web Service Description Language (WSDL) 1.1
  http://www.w3.org/TR/2001/NOTE-wsdl-20010315
- WS-I Basic Profile 1.1
  http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

Interact platform and data model overview

Responsys Interact is a comprehensive on-demand marketing platform with a fully integrated suite of software applications—all built from the ground up on a single-instance, multi-tenant architecture.

Interact Platform

Responsys Interact platform currently offers the following on-demand applications:

- **Interact Campaign™** for multichannel campaign management lets you efficiently create, test, execute, and measure high-volume, highly individualized marketing campaigns across touch-points for compelling ROI.
- **Interact Program™** for dialogue & event-based marketing helps you orchestrate and automate intelligent, customer-driven dialogues at desired moments in the customer lifecycle for more relevant, profitable interactions.
- **Interact Team™** for marketing process management is designed to help you plan, coordinate, and monitor marketing projects and resources for greater marketing efficiency and improved collaboration among geographically distributed marketing teams.
- **Interact Insight™** for predictive analytics and contact optimization uses cutting-edge analytical models to identify your most relevant customer segments and produce contact strategies optimized for each segment.
- **Interact Connect™** for data integration makes it easy to integrate Responsys Interact with your enterprise or marketing information systems to better utilize marketing data and gain a complete view of customers at every interaction point.

Interact Object Data Model

You can use the Interact platform to create and manage a variety of Interact objects to manage your marketing database and execute your marketing campaigns. The Interact object model consists of these types of objects:

- **Programs** let you assemble multi-campaign dialogs.
- **Campaigns** help you execute email campaigns in batch launch or triggered modes.
- **Forms** enable you to collect data via web forms (not currently supported via the Interact API).
- **Documents** consist of re-usable creative content that is available for use in any Campaign or Form.
- **Data objects** enable you to store and use data for a variety of purposes.
  - **Lists and related objects (Filters, Proof Groups, Segmentations)** store recipient audience records and are used primarily for campaign targeting and personalization.
  - **Profile Extension Tables** store additional information for each unique recipient in your profile list table.
  - **Supplemental Tables and related objects (Filters, SQL object, Join objects)** store miscellaneous data that can be used to define a multi-table relational schema for advanced levels of segmentation, targeting and message personalization.
  - **Link Tables** store campaign link tracking information.

The Interact API provides control over many of these objects, allowing client applications to create, change, or remove these objects in a programmatic way to accomplish a variety of marketing automation goals.

Programs

Program objects define multi-step dialogs that involve a variety of campaign messaging and routing rules based on individual profile and behavioral attributes. Creation of an individual Program takes place in a visual, drag-and-drop user interface that is part of the Interact Program module. The Interact API can be used to trigger Custom Events which enter an individual into or affect the individual's routing in a program.

Campaigns

Campaign objects define the basic behavior of an email campaign in terms of audience, message, and settings.

- General properties include name, type (email or mobile), description, categorization, and other fields that identify the campaign.
- Audience selectors include a list, inclusion filters, exclusion filters, and suppression data.
- Message elements include From header, Reply-to header, Subject header, and HTML/Text message documents.
- Settings control tracking options, auto-close behavior, default variables, and so forth.

You can launch a campaign in bulk immediately or schedule it for a later launch. You can also trigger messages from a campaign on demand using form handler rules or program rules.

Forms

Form objects provide functionality for hosting web forms and collecting/processing the data that is submitted. You can use forms to gather customer preferences or for general purpose surveys. Data collected from forms can be merged into a list or supplemental table. Form responses can trigger follow-up emails and custom events that place the responder in a Program dialog.

Documents

Document objects contain the creative content used for campaigns and forms. The two types of the document object are HTML and text. For example, an email campaign usually consists of an HTML and a corresponding text document reference. The campaign handles the display of HTML-only, text-only, or multi-part emails automatically based on the recipient profile. Documents can be re-used across multiple campaigns and forms, copied, edited, and deleted via Interact Campaign.

Lists and related objects

Lists are used to store audience database records—members of your audience might be leads, prospects, customers, contacts, consumers, or visitors, depending on your terminology. The standard set of fields in a list includes:

- Recipient ID (RIID), an internal Responsys Interact-assigned identifier that allows the behavior of individual recipients to be tracked over time.
- Email address, mobile number, postal address, which are standard contact channel fields

- Permission/Opt-in status fields for the various marketing channels (email, mobile, postal)
- Email format preference (HTML or text)
- Derived fields for ISP and domain
- Last modified and created timestamps

In addition, lists can have a number of custom, user-defined fields that you use to maintain a rich audience profile for targeting and personalization purposes.

» Note An account can have any number of lists, but it is recommended that a single central list is used for a given enterprise marketing objective. In some cases, it may make sense to have multiple lists, but use of multiple lists can generate duplicate identities for the same individual audience member.

These are the list-based objects:

- **List filters** are user-defined segments that contain a subset of the members of a list. You can use list filters to include or exclude members from any given campaign launch.
- **List segmentations** are a way of understanding how a list breaks down in terms of a given set of segments. For example, multiple purchasers, one-time purchasers, and non-purchasers.
- **List seeds** store records that share the same schema for a given list, but are used for testing and seeding of campaigns. These records do not represent real members (prospects, customers, and so forth).

Profile extension tables

One or more Profile Extension Tables can be associated with a Profile List. There must be a one-to-one relationship between a record in a Profile Extension Table and its parent Profile List. Profile Extension Tables provide an attractive and efficient way to organize and process audience data. Similar to data in Profile Lists, audience data in Profile Extension Tables can be used for segmentation and targeting in Filters as well as Programs.

Supplemental tables and related objects

As its name indicates, a supplemental table is a collection of database records that supplements a list with additional related information. The connections between a table and a list is made via a *data extraction key*, or key field, that is present in both the table and the list. Because you define the schema for any tables you create, you can use them for a wide variety of purposes, ranging from message personalization and dynamic content to storing form responses and campaign events.

There are several type of table-based data sources: tables, filters on tables, SQL views (on tables and/or lists), and joins on tables.

» Tip When you use tables to extend a list to represent a multi-table relational marketing database (where a variety of queries or joins could be made on the table), be sure to index your tables to reduce the performance impact associated with full table scans on tables being queried or joined.

Link Tables

Link Tables are used to store data about the links that are tracked for a campaign. The schema for a Link Table is fixed and consists of the following fields:

- **LINK_NAME** defines user-friendly name for the link.
- **LINK_URL** defines the destination URL for a tracked link.
- **LINK_CATEGORY** defines a category for links and is available for reporting.
- **EXTERNAL_TRACKING** defines optional parameters that can be appended to the query-string of the destination URL.

Getting started with the Interact API

This section contains general instructions for using the Interact API as well as guidelines and sample code for using the Interact API in a Java or C# application. A Software Development Kit (SDK) with additional sample code and getting started guides for C# and PHP programming languages is also available by contacting the Responsys Support Team at **support@responsys.com**

» Note We assume that you have a basic familiarity with software development, SOAP-based Web Services, and the Responsys Interact platform and user interface.

To get started with the Interact API:

1. Use the Interact Web Services API WSDL to generate supporting code for creating SOAP calls on the Interact API.

   Your development environment or programming language should provide the necessary support for accomplishing this step. The benefit of SOAP/WSDL-based APIs is that most programming languages provide support for managing SOAP requests and responses.

2. Use the Login or LoginWithCertificate calls to establish a session with the Interact Web service.

   Place the session identifier returned by these calls in the SOAP header of all subsequent calls to the Interact API to authenticate the client application.

3. Place a session cookie (JSESSIONID) on the client application after the first successful API call.

   This cookie should be persisted for the duration of the session. Make sure that your client accepts session cookies.

4. Use the available API calls to accomplish a desired goal, including:
   - Data API calls to create, modify or delete individual records.
   - Connect API calls to import or export data in bulk.
   - Campaign API calls to create or modify campaign definitions or launch campaigns.
   - Content API calls to create, modify or delete content documents.

» Note Some Interact API calls have a maximum number of records that can be processed per invocation (**triggerCustomEvent**, plus all data source merge, retrieve, and delete calls). For example, Interact API the limit for calls for triggering campaign messages and merging records into a list is 200 recipients or records. Therefore, you may need to execute these calls in a loop to process additional records during a given client session.

5 If your client application is inactive for longer than two hours and the session identifier becomes invalid, start a new session with a new **Login** call.

6 Use the **Logout** call to end the Interact API session.

You should explicitly log out before attempting a new login call since there is a limit on the number of concurrent sessions you can create for each Interact account.

Java Applications

These are general instructions for getting started with the Interact API from a Java application.

To get started with a Java application:

1 Download the WSDL document. Name the downloaded file **ResponsysWS.wsdl** and place it in your project directory.

Responsys Support will provide the Interact API URLs to you when your account is enabled for Interact API access.

2 Use the **Apache Axis2 WSDL2Java** utility, as described on the **Apache Axis2 web site**, to generate Web Services API stub classes:

- **%AXIS2_HOME%\bin\WSDL2Java -uri ResponsysWS.wsdl -u -d adb -s -p com.rsys.ws.client**
- Assuming the following environment variables are defined:
  - **AXIS2_HOME = C:\axis2-1.3** (or location of the Apache Axis2 Standard Distribution)
  - **AXIS2_LIB = %AXIS2_HOME%\lib**
  - **AXIS2CLASSPATH = %AXIS2_LIB%\axis.jar;%AXIS2_LIB%\jaxrpc.jar;%AXIS2_LIB%\saaj.jar;%AXIS2_LIB%\commons-logging.jar;%AXIS2_LIB%\commons-discovery.jar;%AXIS2_LIB%\wsdl4j.jar**

3 In your Java application, make sure that the generated Interact API stub classes are available to your project build path.

4 Import the following WSDL2Java-generated packages or specific classes needed for your client application calls:

```
import com.rsys.ws.*;
import com.rsys.ws.client.*;
```

5 Instantiate an Interact API service object:

```
service = new ResponsysWSServiceStub("...WS Endpoint URL...");
```

6 Maintain the **JSESSIONID** cookie between requests with the following statement:

```
service._getServiceClient().getOptions().setManageSession(true);
```

7 Instantiate a new Login request object and call the login method of the stub object:

```
Login login = new Login();
login.setUsername("...user...");
```

```
login.setPassword("...pwd...");
LoginResponse response = service.login(login);
```

8 Retrieve the sessionId string from the login result.

9 Submit this sessionId in the SOAP header for all following Interact API calls.

10 Continue with client application logic.

11 End session by logging out when client application task is completed.

Java example

```
import com.rsys.ws.*;
import com.rsys.ws.client.*;
import java.rmi.RemoteException;

public class APITestLoginLogout {
   ResponsysWSServiceStub stub;
   SessionHeader sessionHeader;

   public static void main(String[] args) {
        APITestLoginLogout test = new APITestLoginLogout();
        test.login();
}

private void login() {
      try {
         stub = new ResponsysWSServiceStub("https://...WS Endpoint URL...");
         // maintain session between requests
         stub._getServiceClient().getOptions().setManageSession(true);
// CAUTION: It is important that the user session be maintained. Do no omit preceding step.
         Login login = new Login();
         login.setUsername("...user...");
         login.setPassword("...pwd..");
         LoginResponse response = stub.login(login);
         String sessionId = response.getResult().getSessionId();
         System.out.println ("Login Result = " + sessionId);
         if (sessionId != null) {
            sessionHeader = new SessionHeader();
            sessionHeader.setSessionId(sessionId);
            // Set optional timeout to two minutes
            stub._getServiceClient().getOptions().setTimeOutInMilliSeconds(1000*60*2);
// CAUTION: It is important to set a timeout that is appropriate for the maximum expected duration
    of API calls
            ListFolders listFolders = new ListFolders();
            ListFoldersResponse listFoldersResponse = stub.listFolders(listFolders, sessionHeader);
            FolderResult[] folders = listFoldersResponse.getResult();
            if (folders != null) {
               System.out.println ("Folders length = " + folders.length);
               int i = 0;
               for (FolderResult folder : folders) {
               System.out.println ("Folder Name = " + folder.getName());
                  i++;
               }
            }
            LogoutResponse logoutResponse = stub.logout(new Logout(), sessionHeader);
            boolean loggedOut = logoutResponse.getResult();
            System.out.println("Logout Result = " + loggedOut);
         }
      } catch (AccountFault accountEx) {
            System.out.println ("Ex Code = " + accountEx.getFaultMessage().getExceptionCode());
            System.out.println ("Ex Msg  = " + accountEx.getFaultMessage().getExceptionMessage());
      } catch (UnexpectedErrorFault unexpectedEx) {
            System.out.println ("Ex Code = " + unexpectedEx.getFaultMessage().getExceptionCode());
            System.out.println ("Ex Msg  = " +
    unexpectedEx.getFaultMessage().getExceptionMessage());
      } catch (RemoteException remoteEx) {
            System.out.println ("Ex Msg = " + remoteEx.getMessage());
      }
   }
```

```
}
```

C# Applications

These are general instructions for getting started with the Interact API from a C# application.

To get started with a C# application:

1. Download the WSDL document. Name the downloaded file **ResponsysWS.wsdl**.

   Responsys Support will provide the Web Services API URLs to you when your account is enabled for Web Services API access.

2. Generate the client-side code needed to support your client application's programmatic calls on the Responsys Web service.
   - Open the command window from the Visual Studio menu or include the .NET Framework's bin directory in path environment variable. Type the command **WSDL ResponsysWS.wsdl**
   - Copy the resulting C# file, **ResponsysWSService.cs**, to your project directory.
3. In your C# application, get a handle for the Web Service, and ensure the user session will be maintained. See example provided below.
4. Use the C# compiler to create an executable named fileName.exe, where fileName is the .CS file that contains the Main() method.

   ```
   csc *.cs
   ```
5. Be sure that **csc.exe** is in your path, usually:

   C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxx\)

C# example

```
namespace WSCSharpClient {
   using System;
   using System.Net;
   using System.IO;
   using System.Xml;
   using System.Web.Services.Protocols;

   class TestResponsysWS {
      ResponsysWSService stub;
      bool loggedIn = false;
      SessionHeader sessionHeader;

      private bool login() {
         bool result = false;
         try {
            string url = "... WS Endpoint URL ...";
            Console.WriteLine("Web Services URL = " + url);

            string username = "sjo";
            string password = "sjo";

            stub = new ResponsysWSService();
            stub.CookieContainer = new CookieContainer();
// Caution: It is important that the user session be maintained, so do not omit the preceding step.
            stub.Url = url;
            // Call the login method
            LoginResult loginResult = stub.login(username, password);
            string sessionId = loginResult.sessionId;

            if (sessionId != null) {
               // Create the sessionHeader object and set it to the stub.
               // The sessionHeader is passed to every other API call after the login.
```

```
                sessionHeader = new SessionHeader();
                sessionHeader.sessionId = sessionId;
                stub.SessionHeaderValue = sessionHeader;
// Caution: It is important to set a sessionHeader object to the stub as it is used in all the
    subsequent calls.
                sop("Setting the Client Timeout to 2 minutes");

                // Set timeout
                stub.Timeout = 1000 * 60 * 2;
// Caution: It is important to set a timeout that is appropriate for the maximum expected duration
    of API calls.
                loggedIn = true;
                result = true;
            }
        } catch  (System.Web.Services.Protocols.SoapException e) {
            Console.WriteLine("SoapException in login : " + e.Message);
            Console.WriteLine("SoapException in login : " + e.Detail.InnerText);
        } catch  (Exception e) {
            Console.WriteLine("Exception in login : " + e.Message);
        }
        return result;
    }
  }
}
```

Important .NET WSDL edits required

If you are using the Microsoft .NET WSDL, you must make a correction to the **RecordData** element in the **ResponsysWS.wsdl** file. This element contains an array of Record elements, each of which contains an array of Strings.

However, the Microsoft .NET **wsdl.exe** has a defect that affects arrays inside of other arrays. It creates the **recordsField** as a two-dimensional string array instead of an array of Record class. Furthermore, the Record class is not created at all in the **ResponsysWSService.cs** class. You can fix this by editing the **ResponsysWSService.cs** class to create a **Record** class and changing the two-dimensional string array in the **RecordData** class to an array of **Record** objects.

To make required .NET WSDL edits to the ResponsysWSService.cs class:

1 Create the following **Record** class.

```
/// <remarks/>
  [System.CodeDom.Compiler.GeneratedCodeAttribute("wsdl", "2.0.50727.42")]
  [System.SerializableAttribute()]
  [System.Diagnostics.DebuggerStepThroughAttribute()]
  [System.ComponentModel.DesignerCategoryAttribute("code")]
  [System.Xml.Serialization.XmlTypeAttribute(Namespace="urn:ws.rsys.com")]
  public partial class Record {

    private string[] fieldValuesField;

    /// <remarks/>
    [System.Xml.Serialization.XmlElementAttribute("fieldValues", IsNullable=true)]
    public string[] fieldValues {
      get {
        return this.fieldValuesField;
      }
      set {
        this.fieldValuesField = value;
      }
    }
  }
```

2 Change the **string[][] recordsField** in **RecordData** class to **Record[] recordsField** by replacing the contents of the **RecordData** class with this:

```
/// <remarks/>
    [System.CodeDom.Compiler.GeneratedCodeAttribute("wsdl", "2.0.50727.42")]
    [System.SerializableAttribute()]
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Xml.Serialization.XmlTypeAttribute(Namespace="urn:ws.rsys.com")]
    public partial class RecordData {
        private string[] fieldNamesField;
        private Record[] recordsField;
        /// <remarks/>
        [System.Xml.Serialization.XmlElementAttribute("fieldNames", IsNullable=true)]
        public string[] fieldNames {
            get {
                return this.fieldNamesField;
            }
            set {
                this.fieldNamesField = value;
            }
        }
        /// <remarks/>
        [System.Xml.Serialization.XmlElementAttribute("records", IsNullable=true)]
        public Record[] records {
            get {
                return this.recordsField;
            }
            set {
                this.recordsField = value;
            }
        }
    }
```

Interact API Calls, Types, and Objects

The Interact API calls are divided into these categories:

The Interact API also contains standard primitive types—**boolean**, **string**, **int and long**, and **dateTime**—as well as a collection of objects to be used with API calls.

Session Management API calls

The Session Management API calls are:

- Login
- Logout
- LoginWithCertificate
- AuthenticateServer

Login

Syntax:

LoginResult = service.login(string username, string password)

Usage

The first step for any client application is to establish a login session. This can be achieved using the **login** call.

When a client application invokes the **login** call, it passes a **username** and **password** as user credentials. Upon receiving the client application login request, the WS API authenticates these credentials, and returns a **LoginResult** object. This object can be inspected to retrieve a session token that is required for use in all subsequent API calls. After successfully completing the **login** call and retrieving the session token, a client application needs to set this session token in the SOAP header for subsequent calls as a means of authentication.

Session tokens expire automatically after two hours of inactivity. Client applications that make infrequent login calls should make explicit logout calls to prevent the accumulation of unnecessary open sessions. A limit is placed on the number of concurrent API sessions that an account can initiate. It is important to properly manage API sessions to avoid exceeding this limit. If the limit is reached, an error message will be returned, stating that the allowed number of concurrent sessions has been exceeded.

A JSESSIONID cookie is also set on the client application with the response from the login call. This cookie must be persisted for use in subsequent API calls in the session.

» Note If you are using either Axis2, C# or any other .Net language, the JSESSIONID is automatically captured and sent in subsequent requests. However, if you are not using one of these languages, you must capture the JSESSIONID and Path from the login response HTTP Headers and set them in a cookie in the HTTP headers of all subsequent requests until you log out. This will prevent errors.

Example

```
HTTP/1.1 200 OK
Date: Tue, 16 Nov 2010 14:52:14 GMT
Set-Cookie: JSESSIONID=C1DC1654EE6BBEEBE94043EE4D006F59.tmws2; Path=/
   tmws
Content-Type: text/xml;charset=UTF-8
Connection: close
```

Transfer-Encoding: chunked

Request Arguments

Name	Type	Description
username	string	User name for the Responsys Interact account
password	string	Password for the specified user

Response

The login call returns a **LoginResult** object, which has the following property:

Name	Type	Description
sessionId	string	Unique Session ID associated with this session. Your client application needs to set this value in the session header of subsequent API calls.

Logout

Syntax

boolean = service.logout()

Usage

Use the logout call to end an API session. The last step for any client application is to end a session by logging out. Note that sessions are terminated automatically after two hours of inactivity.

Request Arguments

None

Response

Name	Type	Description
result	boolean	Flag representing the success of a request to end the API session.

LoginWithCertificate

Syntax

LoginResult = service.loginWithCertificate(byte[] encryptedServerChallenge)

Usage

Use the loginWithCertificate call to establish a login session. This can be achieved using either the login or loginWithCertificate calls. The difference is that the authentication for the login call is based on use of password whereas the authentication for the loginWithCertificate call is based on the use of a digital certificate in accordance with the X.509 standard for public key infrastructure (PKI). It is available for developers that require the security advantages of PKI over password-based authentication.

To develop a client application with this call, the Interact account administrator must log into the Interact user interface, navigate to the admin console, and upload a digital certificate (client user public key) and download the Interact API server digital certificate (server public key). These certificates will be used by the client application to log in with the loginWithCertificate call.

The client application establishes an authenticated session in two steps. First, the client application uses the authenitcateServer call with a user name and client challenge and then receives a server challenge, an encrypted response to the client challenge, and a temporary session ID for this authentication step. The client application confirms that the server is authentic and prepares a response to the server challenge. The second step of the authentication involves calling loginWithCertificate with the response to the server challenge and the temporary session ID placed in the SOAP header.

The Interact API then authenticates these credentials, and returns a LoginResult object. This object can be inspected to retrieve a new session token that is required for use in all subsequent API calls. After successfully completing the loginWithCertificate call and retrieving the session token, a client application needs to set this session token in the SOAP header for subsequent calls as a means of authentication.

Session tokens expire automatically after two hours of inactivity. Client applications that make infrequent login calls should make explicit logout calls to prevent the accumulation of unnecessary open sessions. A limit is placed on the number of concurrent API sessions that an account can initiate. It is important to properly manage API sessions to avoid exceeding this limit. If the limit is reached, an error message will be returned, stating that the allowed number of concurrent sessions has been exceeded.

To use this call:

1. Prepare a client challenge as a byte array.
2. Call authenticateServer with an Interact user name and the client challenge and receive a server challenge, an encrypted response to the client challenge, and a temporary session ID for this authentication process.
3. Validate the encrypted client challenge by decrypting with the server public key. Abort if the server authenticity cannot be confirmed.
4. Prepare a response to the server challenge by encrypting the server challenge with the client private key.

5 Call loginWithCertificate with the encrypted server challenge and the temporary session ID placed in the SOAP header.

6 The Interact API will authenticate the client by decrypting the server challenge with the previously uploaded client public key.

7 Upon successful authentication, the Interact API will respond with a LoginResult object from which a valid session ID can be retrieved for use in all subsequent API calls.

Request Arguments

Name	Type	Description
encryptedServerChallenge	byte[]	Encrypted value of the server challenge. The server challenge is encrypted using the client private key that corresponds to a client public key certificate that was uploaded via the Interact admin console as the means to authenticate Interact API session requests.

Response

This call returns a LoginResult object, which has the following property:

Name	Type	Description
sessionId	string	Unique Session ID associated with this session. Your client application needs to set this value in the session header of subsequent API calls.

AuthenticateServer

Syntax

```
ServerAuthResult = service.authenticateServer(string username, byte[]
   clientChallenge)
```

Usage

Use the authenticateServer call to authenticate the Interact API server and initiate a successful login to the Interact API. The information returned from this API call can be used to successfully log in to the Interact API with the loginWithCertificate call.

A client application can establish an authenticated session in two steps.

1 First, the client application uses the authenticateServer call with a user name and client challenge and then receives a server challenge, an encrypted response to the client challenge, and a temporary session ID for this authentication step. The client application confirms that the server is authentic and prepares a response to the server challenge.

2 The second step of the authentication involves calling loginWithCertificate with the response to the server challenge and the temporary session ID placed in the SOAP header. The login process with authenticateServer and loginWithCertificate is described in more detail under the loginWithCertificate section above.

» **Note** A JSESSIONID cookie is also set on the client application with the response from the authenticateServer call. This cookie must be persisted for use in subsequent API calls in the session.

Request Arguments

Name	Type	Description
username	string	User name for the Interact account of interest.
clientChallenge	byte[]	Client application challenge of the server which is used to confirm the authenticity of the server.

Response

The login call returns a ServerAuthResult object, which has the following properties:

Name	Type	Description
authSessionId	string	Temporary session ID that should be placed in the SOAP header of the subsequent loginWithCertificate call.
encrytpedClientChallenge	byte[]	Response to the client challenge, represented by encrypting the client challenge with the server private key. Client applications should validate server authenticity by decrypting this value with the server public key (available through the Interact user interface admin console).
serverChallenge	byte[]	Server challenge of client application authenticity. This challenge should be encrypted with the client private key and submitted with the loginWithCertificate call to authenticate the client application session.

Folder Management API calls

The Folder Management API calls are:

- CreateContentLibraryFolder
- CreateFolder
- DeleteContentLibraryFolder
- DeleteFolder
- DoesContentLibraryFolderExist
- ListContentLibraryFolders
- ListFolders

CreateContentLibraryFolder

Syntax

HierarchyElement = service.createContentLibraryFolder (String folderName)

Usage

Use the create ContentLibraryFolder call to create a new empty folder in the Content Library.

Request Arguments

Name	Type	Description
folderName	string	The name of the folder to create.

Response

Name	Type	Description
result	HierarchyElement	The content library folder.

CreateFolder

Syntax

boolean = service.createFolder(string folderName)

Usage

Use the createFolder call to create a new empty folder in an Interact account. This call returns a boolean value that indicates the success of the folder creation request.

Request Arguments

Name	Type	Description
folderName	string	Name of the folder to create

Response

Name	Type	Description
result	boolean	Success flag folder creation

DeleteContentLibraryFolder

Syntax

void service.deleteContentLibraryFolder (String folderName)

Usage

Use the deleteContentLibraryFolder call to delete a folder and its contents from the Content Library.

Request Arguments

Name	Type	Description
folderName	string	The name of the folder to delete.

Response

Name	Type	Description
void	N/A	Delete content library folder.

DeleteFolder

Syntax

boolean = service.deleteFolder(string folderName)

Usage

Use the deleteFolder call to delete a folder and its contents from an Interact account.

Request Arguments

Name	Type	Description
folderName	string	Name of folder to delete

Response

Name	Type	Description
result	boolean	Success flag for deletion of folder

DoesContentLibraryFolderExist

Syntax

boolean = service.doesContentLibraryFolderExist (String path)

Usage

Use the doesContentLibraryFolderExist call to check whether a specific folder exists in the Content Library.

Request Arguments

Name	Type	Description
path	string	The name of the folder to check.

Response

Name	Type	Description
result	boolean	True if the folder exists.

ListContentLibraryFolders

Syntax

List<HierarchyElement> = service.listContentLibraryFolders(String startingPath, Boolean showTree)

Usage

Use the listContentLibraryFolders call to retrieve a listing of all Content Library folders.

Request Arguments

Name	Type	Description
startingPath	string	The starting parent folder.
showTree	boolean	True displays the full Content Library folder structure. If startingPath is specified, shows all children in the tree, not only the starting path's immediate child folders.

Response

Name	Type	Description
result	**List<HierarchyElement>**	List of Content Library folders.

ListFolders

Syntax

FolderResult[] = service.listFolders()

Usage

Use the listFolders call to retrieve a listing of all of the folders in an account.

Request Arguments

None

Response

The listFolders call returns an array of FolderResult objects. A FolderResult object has a single property.

Name	Type	Description
name	string	Folder name

List Management API calls

The List Management API calls are:

- MergeListMembers
- MergeListMembersRIID
- DeleteListMembers
- RetrieveListMembers

MergeListMembers

Syntax

MergeResult[] = service.mergeListMembers(InteractObject list, RecordData recordData, ListMergeRule mergeRule)

Usage

Use the mergeListMembers call to insert new members or update existing member fields in a given List. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

» Note Using the OR logical operator will result in an error message.

Request Arguments

Name	Type	Description
list	InteractObject	List object
recordData	RecordData	Array of RecordData objects that contain field and record data
mergeRule	ListMergeRule	Defines the merge rules for how to handle the record data

Response

The MergeResult object that is returned from this call has the following properties:

Name	Type	Description
insertCount	long	Number of records inserted
updateCount	long	Number of records updated
rejectedCount	long	Number of records rejected
totalCount	long	Number of records processed
errorMessage	string	Error message if applicable

MergeListMembersRIID

Syntax

```
RecipientResult [] = mergeListMembersRIID(InteractObject list, RecordData
   recordData, ListMergeRule mergeRule)
```

Usage

Use the mergeListMembersRIID call to insert new members or update existing member fields in a given List. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

» **Note** Using the **OR** logical operator will result in an error message.

Request Arguments

Name	Type	Description
list	InteractObject	List object
recordData	RecordData	Array of RecordData objects that contain field and record data
mergeRule	ListMergeRule	Defines the merge rules for how to handle the record data

Response

The RecipientResult object that is returned from this call has the following properties:

Name	Type	Description
recipientId	long	Identifier of the record.
errorMessage	string	Error message if applicable

DeleteListMembers

Syntax

```
DeleteResult[] = service.deleteListMembers(InteractObject list, QueryColumn
    queryColumn, string[] idsToDelete)
```

Usage

Use the deleteListMembers call to delete members from a List by matching on RIID, CUSTOMER_ID, EMAIL_ADDRESS, or MOBILE_NUMBER fields. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
list	InteractObject	List object
queryColumn	QueryColumn	One value from the QueryColumn list of RIID, CUSTOMER_ID, EMAIL_ADDRESS, or MOBILE_NUMBER
idsToDelete	string[]	Values for the specified QueryColumn to match for deletion from the List.

Response

The DeleteResult that is returned from this call has the following properties:

Name	Type	Description
id	string	Identifier of the record that was deleted. The identifier value corresponds to the value of the queryColumn that was matched for the deleted record.
success	boolean	Flag indicating whether deletion request was successfully processed
errorMessage	string	Error message if applicable

RetrieveListMembers

Syntax

RetrieveResult = service.retrieveListMembers(InteractObject list, QueryColumn queryColumn, string[] fieldList, string[] idsToRetrieve)

Usage

Use the retrieveListMembers call to retrieve fields for individual List members. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
List	InteractObject	List object
queryColumn	QueryColumn	One value from the QueryColumn match options: RIID, CUSTOMER_ID, EMAIL_ADDRESS, or MOBILE_NUMBER.
fieldList	string[]	Fields to retrieve from List member record.
idsToRetrieve	string[]	Values for the specified QueryColumn to match for retrieval from the List

Response

The RecordData object that is returned from this call has the following properties:

Name	Type	Description
fieldNames	string[]	String array the names of fields returned
records	Record[]	Record array of the record data returned. The order of the field values returned for each Record is the same order as the fieldNames array.

Table Management API calls

The Table Management calls are:

- CreateTable
- CreateTableWithPK
- DeleteProfileExtensionMembers
- DeleteTable
- MergeIntoProfileExtension
- MergeTableRecords
- MergeTableRecordsWithPK
- DeleteTableRecords
- RetrieveTableRecords
- RetrieveProfileExtensionRecords
- TruncateTable

CreateTable

Syntax

boolean = service.createTable(InteractObject table, Field[] fields)

Usage

Use the createTable call to create a table with a user-defined schema. Tables can be used in a variety of ways, ranging from use as a source of supplemental data to a List, related to the List through *data extraction key* field(s), as a lookup table for generating dynamic content in a campaign message, or as a form response table.

Request Arguments

Name	Type	Description
table	InteractObject	Table object
fields	Field []	Fields to create. You can also specify data extraction keys via the fields array.

Response

Name	Type	Description
result	boolean	Success flag for table creation request

CreateTableWithPK

Syntax

boolean = service.createTableWithPK (InteractObject table, Field[] fields, String[] primaryKeys)

Usage

Use this function to create a supplemental table with a user-defined schema and designate a set of one or more fields as the table's primary key.

Request Arguments

Name	Type	Description
table	InteractObject	Table object
fields	Field[]	Fields to create. You can also specify data extraction keys via the fields array
primaryKeys	String[]	An array containing the names of fields that define the primary key of the table

Response

Name	Type	Description
result	boolean	Success flag for table creation request

DeleteProfileExtensionMembers

Syntax

DeleteResult[] = service.deleteProfileExtensionMembers (InteractObject listExt, QueryColumn queryColumn, string[] idsToDelete)

Usage

Use the deleteProfileExtensionMembers call to delete members from a Profile Extension Table by matching on RIID, CUSTOMER_ID, EMAIL_ADDRESS, or MOBILE_NUMBER fields from the parent list. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
listExtension	InteractObject	Profile Extension object
queryColumn	QueryColumn	One of the following values from the QueryColumn list: RIID CUSTOMER_ID EMAIL_ADDRESS MOBILE_NUMBER
idsToDelete	String[]	Values for the specified QueryColumn to match

Response

The DeleteResult that is returned from this call has the following properties:

Name	Type	Description
id	String	Identifier of the record that was deleted. The identifier value corresponds to the value of the queryColumn that was matched for the deleted record.
success	boolean	Flag indicating whether the deletion request was successfully processed
errorMessage	String	Error message, if applicable

DeleteTable

Syntax

boolean = service.deleteTable(InteractObject table)

Usage

Use the deleteTable call to delete a table from your account.

Request Arguments

Name	Type	Description
table	InteractObject	Table object

Response

Name	Type	Description
result	boolean	Success flag for table deletion request

MergeIntoProfileExtension

Syntax

RecipientResult[] = service.mergeIntoProfileExtension(InteractObject profileExtension, RecordData recordData, QueryColumn queryColumn, boolean insertOnNoMatch, UpdateOnMatch updateOnMatch)

Usage

Use the MergeIntoProfileExtension call to insert or update records in a Profile Extension Table. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
profileExtension	InteractObject	profileExtension contains two fields: String folderName & String objectName. The objectName in this case is the name of the Profile Extension Table.
recordData	RecordData	Array of RecordData objects that contain field and record data.
matchColumn	QueryColumn	Column for which a match attempt should be attempted as part of the merge operation.
insertOnNoMatch	boolean	Indicates what should be done for records where a match is not found (true = insert / false = no insert).
updateOnMatch	UpdateOnMatch	Controls how the existing record should be updated.

Response

A RecipientResult object having the following properties is returned from this call:

Name	Type	Description
recipientId	long	Identifier of the record
errorMessage	string	Error message if applicable

MergeTableRecords

Syntax

MergeResult[] = service.mergeTableRecords(InteractObject table, RecordData records, string[] matchColumnNames)

Usage

Use the mergeTableRecords call to insert or update records in a table. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
table	InteractObject	Table object
records	RecordData	RecordData object that contains field and record data for the merge operation

Name	Type	Description
matchColumnNames	string[]	Column for which a match attempt should be attempted as part of the merge operation. If there is a match for with an existing record, that record will be updated. If there is not a match, then a new record is inserted. Currently only a single match column can be used. So the length of the matchColumnNames string array is limited to one. Future versions of the API will support matches on multiple columns.

Response

A MergeResult object having the following properties is returned from this call:

Name	Type	Description
insertCount	long	Number of records inserted
updateCount	long	Number of records updated
rejectedCount	long	Number of records rejected
totalCount	long	Number of records processed
errorMessage	string	Error message if applicable

MergeTableRecordsWithPK

Syntax

MergeResult[] = service.mergeTableRecordsWithPK (InteractObject table, RecordData recordData, boolean insertOnNoMatch, UpdateOnMatch updateOnMatch)

Usage

Use this function to update or insert data into a supplemental table that has a primary key.

Request Arguments

Name	Type	Description
table	InteractObject	Table object
recordData	RecordData	Array of RecordData objects that contain field and record data
insertOnNoMatch	boolean	Indicates what should be done for records where a match is not found (true = insert / false = no insert).

Name	Type	Description
updateOnMatch	UpdateOnMatch	Controls how the existing record should be updated.

» Note This API call doesn't have a match column because the primary key of the table is used as the match column. If a primary key is not defined for the table, an error message is returned.

Response

A MergeResult object having the following properties is returned from this call:

Name	Type	Description
insertCount	long	Number of records inserted
updateCount	long	Number of records updated
rejectedCount	long	Number of records rejected
totalCount	long	Number of records processed
errorMessage	string	Error message if applicable

DeleteTableRecords

Syntax

```
DeleteResult[] = service.deleteTableRecords(InteractObject table, string
    queryColumn, string[] idsToDelete)
```

Usage

Use the deleteTableRecords call to delete records from a table. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
table	InteractObject	Table object
queryColumn	string	Column for which a match attempt should be attempted as part of the delete operation. If there is a match for with an existing record, that record will be deleted. If there is no match, then no record will be deleted and the success flag of the corresponding DeleteResult object will be set to false.
idsToDelete	string[]	Values for the specified QueryColumn to match for deletion from the table.

Response

The DeleteResult that is returned from this call has the following properties:

Name	Type	Description
id	string	Identifier of the record that was deleted. This identifier corresponds to the queryColumn value of the record.
success	boolean	Flag indicating whether the deletion request was successfully processed
errorMessage	string	Error message, if applicable

RetrieveTableRecords

Syntax

```
RetrieveResult = service.retrieveTableRecords(InteractObject table, string
   queryColumn, string[] fieldList, string[] idsToRetrieve)
```

Usage

Use the retrieveTableRecords call to retrieve fields for individual table records. Individual invocations of this API call are limited to 200 records. If you need to process more than 200 records, you should place multiple invocations.

Request Arguments

Name	Type	Description
table	InteractObject	Table object
queryColumn	string	Column name that will be queried for the idsToRetrieve values provided in this call. An index should be placed on the column used for retrieve queries.
fieldList	string[]	Fields to retrieve from table record.
idsToRetrieve	string[]	Values for the specified QueryColumn to match for retrieval from the List

Response

The RecordData object that is returned from this call has the following properties:

Name	Type	Description
fieldnames	string[]	String array the names of fields returned
Records	Record[]	Record array of the record data returned. The order of the field values returned for each Record is the same order as the fieldNames array.

RetrieveProfileExtensionRecords

Syntax

RetrieveResult = service.retrieveProfileExtensionRecords (InteractObject listExtension, QueryColumn queryColumn, String[] fieldList, String[] idsToRetrieve)

Usage

Use the retrieveProfileExtensionRecords call to retrieve fields for individual table records in a profile extension table (PET).

Request Arguments

Name	Type	Description
InteractObject	listExtension	Profile extension table object
QueryColumn	queryColumn	Column name that will be queried for the idsToRetrieve values provided in this call. » Note Only the RIID column is supported at this time.
fieldList	string[]	Fields to retrieve from table record.
idsToRetrieve	string[]	Values for the specified QueryColumn to be matched when retrieving records from the table.

Response

The RecordData object that is returned from this call has the following properties:

Name	Type	Description
fieldnames	string[]	String array the names of fields returned
Records	Record[]	Record array of the record data returned. The order of the field values returned for each Record is the same order as the fieldNames array.

TruncateTable

Syntax

boolean = service.truncateTable(InteractObject table)

Usage

Use the truncateTable call to remove all the records from a table.

Request Arguments

Name	Type	Description
folderName	string	Name of folder containing table to truncate.
tableName	string	Name of table to truncate.

Response

Name	Type	Description
result	boolean	Success flag for truncating a table.

Content Management API calls

The Content Management calls are:

- CopyContentLibraryItem
- CreateContentLibraryItem
- CreateDocument
- DeleteContentLibraryItem
- DeleteDocument
- GetContentLibraryItem
- GetDocumentContent
- GetDocumentImages
- MoveContentLibraryItem
- SetDocumentContent
- SetDocumentImages
- UpdateContentLibraryItem

CopyContentLibraryItem

Syntax

boolean = service.copyContentLibraryItem (String srcPath, String dstPath)

Usage

Use the copyContentLibraryItemcall to copy a Content Library item to a new location.

Request Arguments

Name	Type	Description
srcPath	string	Location from which to copy.
dstPath	string	Location to which to copy.

Response

Name	Type	Description
result	boolean	True if the item was copied.

CreateContentLibraryItem

Syntax

boolean = service.createContentLibraryItem (String folderName, String objectName, ItemData itemData)

Usage

Use the createContentLibraryItem call to create an item in the Content Library.

Request Arguments

Name	Type	Description
folderName	string	Folder in which to create the item.
objectName	string	Name of the item to create.
itemData	ItemData	The filesto upload.

Response

Name	Type	Description
result	boolean	True if the item was created.

CreateDocument

Syntax

boolean = service.createDocument(String folderName, String documentName, String content, String charset)

Usage

Use the createDocument call to create new documents in an Interact Account. If the document contains relative references to images that should be hosted by Interact, then the setDocumentImages call should be made to upload the corresponding image files.

For documents in the Content Library, a full Content Library folder path is required.

Request Arguments

Name	Type	Description
folderName	string	Folder in which to create the document.
documentName	string	Name of the document to create.
content	string	Text content of the document (including markup for HTML content).
charaset	string	Character set of document content.

Response

Name	Type	Description
result	boolean	Flag indicating success of create document request.

DeleteContentLibraryItem

Syntax

boolean = service.deleteContentLibraryItem (String folderName, String objectName)

Usage

Use the deleteContentLibraryItem call to delete an item from the Content Library.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the item to delete.
objectName	string	Name of the item to delete.

Response

Name	Type	Description
result	boolean	True if the item was deleted.

DeleteDocument

Syntax

boolean = service.deleteDocument(String folderName, String documentName)

Usage

Use the deleteDocument call to delete a document from an Interact account.

For documents in the Content Library, a full Content Library folder path is required.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the document to delete.
documentName	string	Name of the document to delete.

Response

Name	Type	Description
Result	boolean	Flag indicating success of delete document request.

GetContentLibraryItem

Syntax

ItemData = service.getContentLibraryItem (String folderName, String objectName)

Usage

Use the getContentLibraryItem call to retrieve the content of a Content Library item.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the item to retrieve.
objectName	string	Name of the item to retrieve.

Response

Name	Type	Description
result	ItemData	The binary data.

GetDocumentContent

Syntax

ContentResult = service.getDocumentContent(String folderName, String documentName)

Usage

Use the getDocumentContent call to obtain the text/markup content of a document object.

For documents in the Content Library, a full Content Library folder path is required.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the document.
documentName	string	Name of the document.

Response

A ContentResult object is returned. This object has the following properties.

Name	Type	Description
Content	string	Text content of document.
Format	ContentFormat	Type of content: HTML or TEXT.
characterEncoding	CharacterEncoding	Character set of document content.

GetDocumentImages

Syntax

ImageData[] = service.getDocumentImages(String folderName, String documentName)

Usage

Use the getDocumentImages call to retrieve the image file content for a document object.

For documents in the Content Library, a full Content Library folder path is required.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the document.
documentName	string	Name of the document.

Response

Name	Type	Description
Result	ImageData[]	Array of ImageData objects corresponding to each image file to be uploaded. The ImageData object has a string property for the image name and a base64binary representation of the image content.

MoveContentLibraryItem

Syntax

boolean = service.moveContentLibraryItem (String srcPath, String dstPath)

Usage

Use the moveContentLibraryItem call to move a Content Libraray item to a new location.

Request Arguments

Name	Type	Description
srcPath	string	Location from which to move.
dstPath	string	Location to which to move.

Response

Name	Type	Description
result	boolean	True if the item was moved.

SetDocumentContent

Syntax

boolean = service.setDocumentContent(String folderName, String documentName, String content)

Usage

Use the setDocumentContent call to change the text content of a document object.

For documents in the Content Library, a full Content Library folder path is required.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the document.
documentName	string	Name of the document.
content	string	Text content to set for existing document.

Response

Name	Type	Description
result	boolean	Flag indicating success of set content request.

SetDocumentImages

Syntax

CommonResult = service.setDocumentImages(String folderName, String documentName,ImageData[] imageData)

Usage

Use the setDocumentImages call to upload images files for a document.

For documents in the Content Library, a full Content Library folder path is required.

Request Arguments

Name	Type	Description
folderName	string	Folder containing the document.
documentName	string	Name of the document.
imageData	ImageData[]	Array of ImageData objects corresponding to each image file to be uploaded. The ImageData object has a string property for the image name and a base64binary representation of the image content.

Response

Name	Type	Description
result	boolean	Flag indicating success of set images request.

UpdateContentLibraryItem

Syntax

boolean = service.updateContentLibraryItem (String folderName, String objectName, ItemData itemData)

Usage

Use the updateContentLibraryItem call to update a jpg, gif, png, pdf, tif, or swf item in the Content Library.

Request Arguments

Name	Type	Description
folderName	string	Folder contatining the item to update.
objectName	string	Name of the item to update.
itemData	ItemData	The data to update.

Response

Name	Type	Description
result	boolean	True if the item was updated.

Campaign Management API calls

The Campaign Management calls are:

- GetLaunchStatus
- LaunchCampaign
- MergeTriggerEmail
- ScheduleCampaignLaunch
- TriggerCustomEvent
- TriggerCampaignMessage

GetLaunchStatus

Syntax

LaunchStatusResult[] = service.getLaunchStatus(long[] launchIds)

Usage

Use the getLaunchStatus call to retrieve launch information for one or more launch identifiers.

Request Arguments

Name	Type	Description
launchIds	long[]	An array of launch identifiers which may have been retrieved and persisted by several possible previous API calls in the client application.

Response

An array of LaunchStatusResult objects is returned. The LaunchStatusResult object has the following properties:

Name	Type	Description
launchId	long	Launch identifier
launchState	string	Launch State: ■ PENDING ■ LAUNCHING ■ USER_PAUSE ■ USER_ABORT ■ SYSTEM_PAUSE ■ SYSTEM_ABORT ■ DONE
launchType	string	Launch Type: ■ PROOF ■ STANDARD
launchDate	dateTime	Timestamp for when launch was initiated.
campaign	InteractObject	Campaign object

LaunchCampaign

Syntax

LaunchResult = service.launchCampaign(InteractObject campaign, ProofLaunchOptions proofLaunchOptions, LaunchPreferences launchPreferences)

Usage

Use the launchCampaign to immediately initiate a campaign launch. A numeric launch identifier is returned from this call and allows for the monitoring of the launch status.

Request Arguments

Name	Type	Description
Campaign	InteractObject	Campaign object reference
proofLaunch Options	ProofLaunch Options	For proof launches, specify several options as properties of the ProofLaunchOptions object: **proofEmailAddress**: comma separated email address(es) to send proof launches to **proofLaunchType**: ■ LAUNCH_TO_ADDRESS ■ LAUNCH_TO_PROOFLIST ■ LAUNCH_TO_ADDRESS_USING_PROOFLIST
launch Preferences	Launch Preferences	LaunchPreference object properties include: boolean enableLimit int recipientLimit boolean enableNthSampling int samplingNthSelection int samplingNthInterval int samplingNthOffset boolean enableProgressAlerts string progressEmailAddresses int progressChunk (>999)

Response

Returns a LaunchResult which contains the following properties:

Name	Type	Description
launchId	long	Launch identifier

MergeTriggerEmail

Syntax

TriggerResult[] = service.mergeTriggerEmail(RecordData recordData, ListMergeRule mergeRule, InteractObject campaign, TriggerData[] triggertData)

Usage

Use the mergeTriggerEmail function to merge member(s) into the profile list and subsequently trigger email message(s) to the merged member(s) all in a single call.

Request Arguments

Name	Type	Description
recordData	RecordData	Array of RecordData objects that contain field and record data
mergeRule	ListMergeRule	Defines the merge rules for how to handle the record data
campaign	InteractObject	Campaign name and folder
triggerData	TriggerData[]	An array of TriggerData objects that consists of an OptionalData object array (see below)

Response

The MergeTriggerEmail call returns an array of TriggerResult objects. This object has the following properties:

Name	Type	Description
recipientId	Long	Interact internal recipient ID (RIID_) for the individual to whom the message was sent.
success	Boolean	Success flag for trigger message request.
errorMessage	String	NO_RECIPIENT_FOUND MULTIPLE_RECIPIENTS_FOUND

ScheduleCampaignLaunch

Syntax

boolean = service.scheduleCampaignLaunch(InteractObject campaign, ProofLaunchOptions proofLaunchOptions, LaunchPreferences launchPreferences, dateTime scheduleDate)

Usage

Use the scheduleLaunch call to schedule the launch of a campaign at some future point in time.

Request Arguments

Name	Type	Description
campaign	InteractObject	Campaign object reference
proofLaunch Options	ProofLaunch Options	Leave null for standard launches. For proof launches, specify several options as properties of the ProofLaunchOptions object: **proofEmailAddress**: comma separated email address(es) to send proof launches to **proofLaunchType**: LAUNCH_TO_ADDRESS LAUNCH_TO_PROOFLIST LAUNCH_TO_ADDRESS_USING_PROOFLIST
launch Preferences	Launch Preferences	LaunchPreference object properties include: boolean enableLimit int recipientLimit boolean enableNthSampling int samplingNthSelection int samplingNthInterval int samplingNthOffset boolean enableProgressAlerts string progressEmailAddresses int progressChunk (>999)
scheduleDate	dateTime	Date and time for launch.

Response

Name	Type	Description
result	boolean	Flag for the success of the launch request

TriggerCustomEvent

Syntax

TriggerResult[] = service.triggerCustomEvent(CustomEvent customEvent, RecipientData[] recipientData)

Usage

Use the triggerCustomEvent call to trigger a Custom Event for a recipient. The Interact platform provides Custom Event listeners that will respond to a triggered Custom Event in several possible ways depending on the specific definition and use of Custom Events in your Interact account. Some Custom Events provide an entry point into one or more Interact Programs. Other Custom Events can be used for segmentation purposes. See the Interact platform documentation for more information on the use of Custom Events.

A single triggerCustomEvent request is limited to 200 recipients. If you need to trigger a Custom Event for more than 200 recipients, then you should place multiple triggerCustomEvent requests.

» Note Sending duplicate names in the recipientData would result in an error message.

Request Arguments

Name	Type	Description
customEvent	CustomEvent	The CustomEvent to be triggered. The CustomEvent eventName or eventId property must be specified for this object.
recipientData	RecipientData[]	An array of RecipientData objects that define the recipients for whom a custom event should be triggered. A RecipientData object consists of a Recipient object and an OptionalData object array. At least one of the following List member identifiers should be provided in the Recipient object (recipientId, emailAddress, customerId, or mobileNumber). If you specify more than one of these values, we process them in this order—recipientId, emailAddress, customerId, or mobileNumber—and we take the first non-null value. For example, if you specify emailAddress and customerId, we only take the emailAddress (unless there are no email addresses).

Response

The triggerCustomEvent call returns an array of TriggerResult objects. The TriggerResult object has the following properties.

Name	Type	Description
recipientId	long	Interact internal recipient ID (RIID_) for the individual to whom the message was sent.
success	boolean	Success flag

Name	Type	Description
errorMessage	string	NO_RECIPIENT_FOUND MULTIPLE_RECIPIENTS_FOUND

TriggerCampaignMessage

Syntax

TriggerResult[] = service.triggerCampaignMessage(InteractObject campaign, RecipientData[] recipientData)

Usage

Use the triggerCampaignMessage call to send email messages to one or more recipients. A single triggerCampaignMessage request is limited to 200 recipients. If you need to trigger to a message to more than 200 recipients, then you should execute multiple triggerCampaignMessage requests.

» Note Sending duplicate names in the recipientData would result in an error message.

Request Arguments

Name	Type	Description
campaign	InteractObject	Campaign name

Name	Type	Description
recipientData	RecipientData[]	An array of RecipientData objects that define the recipients to whom a campaign message should be sent. A RecipientData object consists of a Recipient object and an OptionalData object array. NOTE: This call uses recipientData only to look up a recipient in the list. This means that if you want to change any data, for example, use a specific email format, you must update the user record before making this call. At least one of the following List member identifiers should be provided in the Recipient object (recipientId, emailAddress, customerId, or mobileNumber). If you specify more than one of these values, we process them in this order—recipientId, emailAddress, customerId, or mobileNumber—and we take the first non-null value. For example, if you specify emailAddress and customerId, we only take the emailAddress (unless there are no email addresses). The Recipient object List property is optional for this call since a valid campaign already has a reference to an existing List. The array of OptionalData objects define name/value pairs that can be used for dynamic content in the campaign message template.

Response

The triggerCampaignMessage call returns an array of TriggerResult objects. This object has the following properties.

Name	Type	Description
recipientId	Long	Interact internal recipient ID (RIID_) for the individual to whom the message was sent.
success	Boolean	Success flag for trigger message request.
errorMessage	String	NO_RECIPIENT_FOUND MULTIPLE_RECIPIENTS_FOUND

Interact API Primitive Types

The Interact API uses the primitive data types defined below. These primitive data types are specified in the World Wide Web Consortium's publication "XML Schema Part 2: Data Types" (available at http://www.w3.org/TR/xmlschema-2). Primitive types are used as a standardized way to define, send, receive, and interpret basic data types in the SOAP messages exchanged between client applications and the Interact API.

Type	Description
boolean	Boolean fields have one of these values: true (or 1), or false (or 0).
string	Character string data types contain text data.
int and long	Fields of these types contain integers (long ranges from 9223372036854775807 to -9223372036854775808 and int ranges from 2147483647 to -2147483648.
dateTime	Fields defined as dateTime data types handle date/time values (timestamps). Regular dateTime fields are full timestamps with a precision of one second.

Interact API Objects

These are the Interact API objects you can use.

- CharacterEncoding
- ContentFormat
- CustomEvent
- DeleteResult
- EmailFormat
- Field
- FolderResult
- ImageData
- InteractObject
- LaunchPreferences
- LaunchResult
- ListMergeRule
- LoginResult
- MatchOperator
- MergeResult
- OptionalData
- ProofLaunchOptions
- ProofLaunchOptions
- ProofLaunchType
- QueryColumn
- Recipient
- RecipientData
- RecipientResult
- Record
- RecordData
- ServerAuthResult
- TriggerData
- TriggerResult
- UnsubscribeOption
- UpdateOnMatch

CharacterEncoding

The CharacerEncoding is a string restricted to one of the values listed below.

Type	Values	
string	ISO_8859_1	SJIS
	windows_1257	euc_kr
	ISO_8859_2	koi8_r
	gb2312	ISO_8859_9
	big5	UTF_8
	ISO_8859_7	

ContentFormat

The ContentFormat is a string restricted to one of the values listed below.

Type	Values	
string	HTML\	TEXT

CustomEvent

The CustomEvent object contains information needed for the triggerCustomEvent call.

Name	Type	Description
eventName	string	Name of the Custom Event Type
eventId	long	Identifier for Custom Event Type. Either then nameName or eventId of the Custom Event Type should be specified.
recipients	Recipient[]	Recipients for whom the Custom Event Type will be triggered
optionalData	OptionalData[]	Optional data in the form of an array of name/value pairs that contain additional data for use in downstream custom event processing (either in Interact Program or Behavioral Segmentation).

DeleteResult

The DeleteResult object represents the response from a delete request.

Name	Type	Description
Id	string	Identifier of the record that was deleted.
Success	boolean	Flag indicating whether the deletion request was successfully processed
errorMessage	string	Error message, if applicable

EmailFormat

The EmailFormat is a string restricted to one of the values listed below.

Type	Values	
String	TEXT_FORMAT	MULTIPART_FORMAT
	HTML_FORMAT	NO_FORMAT

Field

The Field object represents a field (or column) in a List or Table.

Name	Type	Description
fieldName	string	Name of field
fieldType	FieldType	Data type of field
Custom	boolean	Flag indicating whether this represents a custom field. This is a read-only variable that is used only in the describeObjects API.
dataExtractionKey	boolean	Flag indicating whether this field is a data extraction key

FieldType

The FieldType is a string restricted to one of the values listed below.

Type	Values	
String	STR500	NUMBER
	STR4000	TIMESTAMP
	INTEGER\	

FolderResult

The Folder object has a single property that defines the name of a folder. In future releases of the Interact WS API, new properties will be added to the Folder object to provide additional folder-related metadata.

Name	Type	Description
Name	string	Folder name

ImageData

The imageData object represents an image file.

Name	Type	Description
imageName	string	Name of image.
image	base64binary	base64binary representation of binary image content.

InteractObject

Name	Type	Description
folderName	string	Name of folder.
objectName	string	Name of object.

LaunchPreferences

The.LaunchPreferences object defines the behavior of the launch.

Name	Type	Description
enableLimit	boolean	Enable limit for launch
recipientLimit	int	Limit launch to a certain number of recipients
enableNthSampling	int	Enable Nth sampling
samplingNthSelection	int	Selection for Nth sampling
samplingNthInterval	int	Interval for Nth sampling
samplingNthOffset	int	Offset for Nth sampling
enableProgressAlerts	boolean	Enable launch progress alerts
progressEmailAddress	string	Email address to sent progress alerts
progressChunk	int	Send progress alerts after the launch of a given number of recipients.

LaunchResult

The LaunchResult object contains information about a campaign launch.

Name	Type	Description
launchId	long	Launch identifier

ListMergeRule

The ListMergeRule object represents the rules by which incoming List records are processed for merging into a List.

Name	Type	Description
insertOnNoMatch	boolean	Indicates what should be done for records where a match is not found (true = insert / false = no insert).
updateOnMatch	UpdateOnMatch	Controls how the existing record should be updated.
matchColumnName1	string	First match column for determining whether an insert or update should occur.
matchColumnName2	string	Second match column for determining whether an insert or update should occur. (optional)
matchOperator	MatchOperator	Controls how the boolean expression involving the match columns is constructed to determine a match between the incoming records and existing records.
optinValue	string	Value of incoming opt-in status data that represents an opt-in status. For example, *1* may represent an opt-in status.
optoutValue	string	Value of incoming opt-out status data that represents an opt-out status. For example, *0* may represent an opt-out status.

Name	Type	Description
defaultPermissionStatus	enum	This value must be specified as either OPTIN or OPTOUT and would be applied to all of the records contained in the API call. If this value is not explicitly specified, then it is set to OPTOUT.
htmlValue	string	Value of incoming preferred email format data. For example, *H* may represent a preference for HTML formatted email.
textValue	string	Value of incoming preferred email format data. For example, *T* may represent a preference for Text formatted email.
rejectRecordIfChannelEmpty	string	String containing comma-separated channel codes that if specified will result in record rejection when the channel address field is null. Channel codes are E, M, P. For example *E,M* would indicate that a record that has a null for Email or Mobile Number value should be rejected.
defaultPermissionStatus	enum	OPTIN, OPTOUT

LoginResult

The LoginResult object has a single property that defines the session ID for a client application session.

Name	Type	Description
sessionId	string	Valid session ID for use in subsequent API calls. This session ID should be placed in the SOAP header for subsequent calls.

MatchOperator

The MatchOperator is a string restricted to one of the values listed below.

Type	Values		
string	NONE	AND	OR

MergeResult

The MergeResult object represents the response from a merge request.

Name	Type	Description
insertCount	long	Number of records inserted
updateCount	long	Number of records updated
rejectedCount	long	Number of records rejected
totalCount	long	Number of records processed
errorMessage	string	Error message if applicable

OptionalData

The OptionalData object contains name/value pair data that can be used in a variety of ways ranging from optional campaign variables to Interact Program enactment variables.

Name	Type	Description
Name	string	Name of variable
Value	string	Value of variable

ProofLaunchOptions

The.ProofLaunchOptions object defines how a proof launch should be conducted.

Name	Type	Description
proofEmailAddress	string	String of comma-separated email addresses
proofLaunchType	ProofLaunchType	Object that defines the nature of the proof launch

ProofLaunchType

The ProofLaunchType is a string restricted to one of the values listed below:

Type	Values
string	LAUNCH_TO_ADDRESS
	LAUNCH_TO_LIST
	LAUNCH_TO_ADDRESS_USING_LIST

QueryColumn

The QueryColumn is a string restricted to one of the values listed below.

Type	Values	
string	RIID	EMAIL_ADDRESS
	CUSTOMER_ID	MOBILE_NUMBER

Recipient

The Recipient object has the following properties. At least one of the Recipient identifiers should be used to uniquely target a recipient: recipientId, customerId, emailAddress, or mobileNumber.

Name	Type	Description
listName	string	Name of list for recipient
recipienId	long	Internal Interact ID (RIID_) for recipient
customerId	string	Externally defined customer ID
emailAddress	string	Email address
mobileNumber	string	Mobile number
emailFormat	EmailFormat	Format of message to deliver to the recipient (optional)

RecipientData

The RecipientData object has the following properties. It is used to represent a List member and a number of name/value pair parameters needed for triggering messages or custom events.

Name	Type	Description
recipient	Recipient	Identity of a List member
optionalData	OptionalData[]	Optional name/value pair parameters associated with this List member.

RecipientResult

The RecipientResult object has the following properties. It returns an array of RecipientResult objects that each contain a recipientID and an errorMessage.

Name	Type	Description
recipientId	long	Identifier of the record
errorMessage	string	Error message if applicable

Record

The Record object represents a record of data from a List or Table.

Name	Type	Description
fieldValues	string[]	A string array representing the values of fields in a record.

RecordData

The RecordData object represents a number of records of data from a List or Table.

Name	Type	Description
fieldNames	string[]	An array containing the field names in a record of data
records	Record[]	An array of Record objects which contain data from a List or Table.

ServerAuthResult

Name	Type	Description
authSessionId	string	Temporary session ID that should be placed in the SOAP header of the subsequent loginWithCertificate call.
encryptedClientChallenge	byte[]	Response to the client challenge, represented by encrypting the client challenge with the server private key. Client applications should validate server authenticity by decrypting this value with the server public key (available through the Interact user interface admin console).
serverChallenge	byte[]	Server challenge of client application authenticity. This challenge should be encrypted with the client private key and submitted with the loginWithCertificate call to authenticate the client application session.

TriggerData

The TriggerData object defines an array of optional name/value pairs that can be used for dynamic content in the campaign message.

Name	Type	Description
optionalData	Optional Data[]	Array of OptionalData objects define name/value pairs that can be used for dynamic content in the campaign message.

TriggerResult

The TriggerResult object defines the results from a trigger request for a campaign message or custom event.

Name	Type	Description
recipientId	long	Interact internal recipient ID (RIID_) for the individual to whom the message was sent.
Success	boolean	Success flag
errorMessage	string	NO_RECIPIENT_FOUND, MULTIPLE_RECIPIENTS_FOUND

UnsubscribeOption

The UnsubscribeOption is a string restricted to one of the values listed below.

Type	Values	
String	NO_OPTOUT_BUTTON	OPTOUT_FORM
	OPTOUT_SINGLE_CLICK	

UpdateOnMatch

The UpdateOnMatch is a string restricted to one of the values listed below.

Type	Values	
String	NO_UPDATE	REPLACE_ALL

Sample Code for Handling Exceeded Account Limits

The following sections provide sample code in Java, C#, and PHP for handling the API_LIMIT_EXCEEDED error that is returned when the account limit for calling an API function is exceeded.

Sample Java code

```
private void getJobRunStatus() {
      if (!loggedIn) {
         if (!login()) {
            return;
         }
      }

      String jobRunIdStr = getUserInput("Enter the jobRunId : ");
      try {
         GetJobRunStatus getJobRunStatus = new GetJobRunStatus();
         getJobRunStatus.setJobRunId(Long.parseLong(jobRunIdStr));
         GetJobRunStatusResponse response = stub.getJobRunStatus(getJobRunStatus,
   sessionHeader);
         JobRunStatusResult result = response.getResult();
         if (result != null) {
            System.out.println("****************************************");
            System.out.println("getJobRunStatus is Successful");
            System.out.println("Job Run Status    = " + result.getJobRunStatus());
            System.out.println("Job Run Duration  = " + result.getDurationInSeconds());
            System.out.println("Error Message     = " + result.getErrorMessage());
            System.out.println("Records Added     = " + result.getRecordsAdded());
            System.out.println("Records Processed = " + result.getRecordsProcessed());
            System.out.println("Records Rejected  = " + result.getRecordsRejected());
            System.out.println("Records Updated   = " + result.getRecordsUpdated());
            System.out.println("****************************************");
         }
         else {
            System.out.println("****************************************");
            System.out.println("getJobRunStatus Failed");
            System.out.println("****************************************");
         }
      }
      catch (ConnectFault connectFaultEx) {
         System.out.println("connectFaultEx getJobRunStatus");
         System.out.println("Exception Code = "
               + connectFaultEx.getFaultMessage().getExceptionCode());
         System.out.println("Exception Msg  = "
               + connectFaultEx.getFaultMessage().getExceptionMessage());
      }
      catch (UnexpectedErrorFault unexpectedEx) {
         System.out.println("unexpectedEx getJobRunStatus");
         System.out.println("Exception Code = "
               + unexpectedEx.getFaultMessage().getExceptionCode());
         System.out.println("Exception Msg  = "
               + unexpectedEx.getFaultMessage().getExceptionMessage());
      }
      catch (RemoteException remoteEx) {
         System.out.println("remoteEx getJobRunStatus");
         System.out.println("Exception Msg = " + remoteEx.getMessage());
         if (remoteEx instanceof AxisFault) {
            AxisFault axisFault = (AxisFault) remoteEx;
```

```
            System.out.println("Fault detail element = "
                  + axisFault.getFaultDetailElement().getText());
         }
         if ("API_LIMIT_EXCEEDED".equalsIgnoreCase(remoteEx.getMessage())) {
            retryDelay();
            getJobRunStatus();
         }
      }
   }

   private void retryDelay() {
      int i = 0;
      while (i < 60) { //60 seconds delay
         try {
            System.out.print(". ");
            Thread.sleep(1000);
            i++;
         }
         catch (InterruptedException ex) {
         }
      }
   }
```

Sample C# code

```
private void getConnectJobRunStatus() {
        try {
           String jobRunIdStr = jobRunIdInput;
           //getUserInput("Enter the jobRunId: ");
           long jobID = long.Parse(jobRunIdStr);
           JobRunStatusResult result = stub.getJobRunStatus(jobID);
           if (result != null) {
              Console.WriteLine("****************************************");
              Console.WriteLine("getJobRunStatus is Successful");
              Console.WriteLine("Job Run Status    = " + result.jobRunStatus);
              Console.WriteLine("Job Run Duration  = " + result.durationInSeconds);
              Console.WriteLine("Error Message     = " + result.errorMessage);
              Console.WriteLine("Records Added     = " + result.recordsAdded);
              Console.WriteLine("Records Processed = " + result.recordsProcessed);
              Console.WriteLine("Records Rejected  = " + result.recordsRejected);
              Console.WriteLine("Records Updated   = " + result.recordsUpdated);
              Console.WriteLine("****************************************");
           }
           else {
              Console.WriteLine("****************************************");
              Console.WriteLine("getJobRunStatus Failed");
              Console.WriteLine("****************************************");
           }
        }
        catch (System.Web.Services.Protocols.SoapException e) {
           Console.WriteLine("SoapException in getJobRunStatus : " + e.Message);
           Console.WriteLine("SoapException in getJobRunStatus : " + e.Detail.InnerText);
           if ("API_LIMIT_EXCEEDED".Equals(e.Message)) {
              Console.WriteLine("The API Limit has Exceeded");
              Thread.Sleep(60000);//need to add using System.Threading;
              getConnectJobRunStatus();
           }
        }
        catch (Exception e) {
           Console.WriteLine("Exception in getJobRunStatus : " + e.Message);
```

```
        }
    }
```

Sample PHP code

```
function mGetJobRunStatus($client,$jobID) {
    try {
        $getJobRunStatusResult = $client->getJobRunStatus (array('jobRunId'=>$jobID));
        print('<pre>');
        print_r($getJobRunStatusResult);
        print('</pre>');
    }
    catch(SoapFault $err) {
        if($err->faultstring=='ConnectFault') {
            print "<br>ConnectFault Error";
            print "<br>Exception Message Detail: ".$err->detail->ConnectFault-
     >exceptionMessage."<br>";
        }
        else if($err->faultstring=='API_LIMIT_EXCEEDED') {
            print "<br>API LIMIT EXCEEDED";
            print "<br>Exception Message Detail: ".$err->detail."<br>";
            sleep(60); //60 secs sleep
            mGetJobRunStatus($client,$jobID);
        }
        else {
            print "Other Exception Error: ".$err->getMessage()."\n";
        }
    }
}
```