Marketing. Well Executed.
Email / Mobile / Social / Web

responsys

Interact API Developer Guide

Release 6.11

responsys.com

Contents

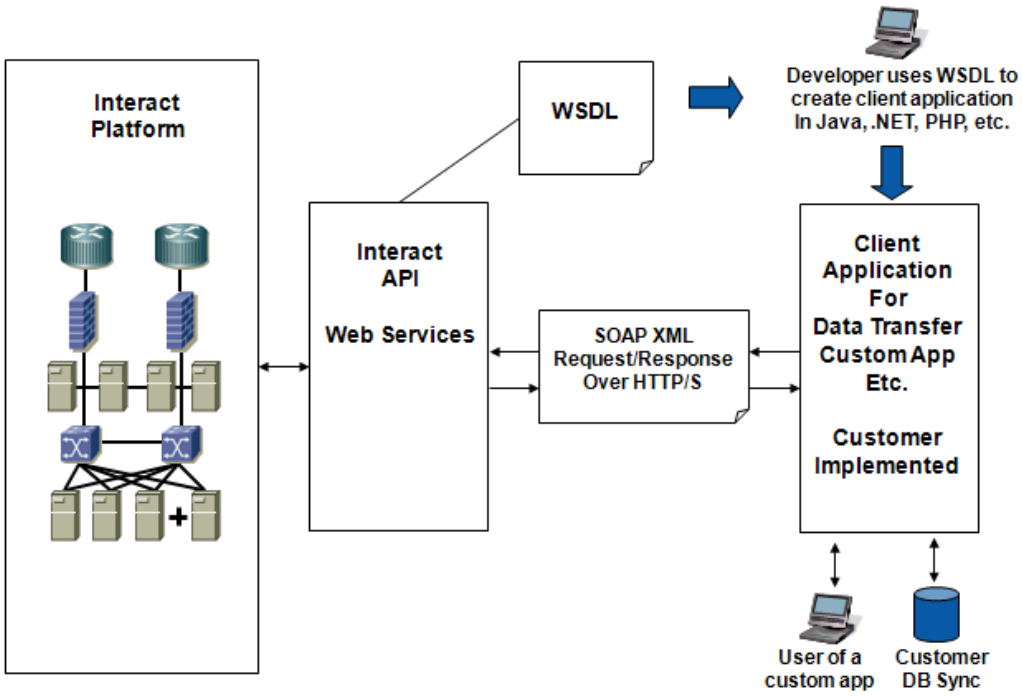
Interact API

The Interact Web Services Application Programming Interface (Interact API) gives your developers 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 lifecycle 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

A conceptual diagram that describes the use of the Interact API is shown below.



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 a subset of the functionality of the Interact user interface and platform as outlined below.

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 SupplementalTables
- Retrieve records from Lists and Supplemental 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

What's New in the Interact API for Interact Suite Release 6.11

Added API Calls

MergeListMembersRIID is an extension to the **MergeListMembers** call and performs a similar function. However, MergeListMembersRIID returns the recipientID of each list member in the response. For applications which already use MergeListMembers, there will be no impact, as that call will continue to work as normal.

Interact API URLs

When your account is enabled for access to the Interact API, the Responsys Support team will provide you with the Web Services URLs needed to develop your projects. Web Services URLs for the Interact 2 pod will be provided when your API account is activated.

Development Environments

The Interact API works with modern SOAP development environments, including, but not limited to, Visual Studio .NET 2005 and Apache Axis. Development platforms vary in their SOAP implementations. Implementation differences in certain development platforms 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 higher.

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-attempts.

Backward Compatibility

Responsys supports backward compatibility as new versions of the Interact API are released. The Interact API is backward compatible in 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. Specifically, the Interact API endpoint URL is versioned so that different Interact API versions have different endpoint URLs. Your applications will continue to work with the Interact API endpoint URLs of previous releases and you will have the opportunity to migrate your client applications to newer Interact API version endpoint URLs to leverage 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 the following 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

Since 1998, Responsys has been a technology leader and innovator in the marketing industry, delivering high-quality campaign management software as a service over the Internet. Best known as the industry's #1-ranked email marketing solution, Responsys Interact® today 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

Currently, the Responsys Interact platform offers the following on-demand applications:

Interact Campaign™ for multichannel campaign management: Efficiently create, test, execute, and measure high-volume, highly individualized marketing campaigns across touchpoints for compelling ROI.

Interact Program™ for dialogue & event-based marketing: 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: 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: Use cutting-edge analytical models to identify your most relevant customer segments and produce contact strategies optimized for each segment.

Interact Connect™ for data integration: Integrate Responsys Interact with your enterprise or marketing information systems for easier leverage of marketing data and a complete view of customers at every interaction point.

Interact Object Data Model

Users of the Interact platform create and manage a variety of Interact objects to manage their marketing database and execute their marketing campaigns. The Interact object model consists of the following types of objects.

- Programs – Allow users to assemble multi-campaign dialogs.
- Campaigns – Allow users to execute email campaigns in batch launch or triggered modes
- Forms – Allow users 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 – Allow users 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.
    - 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 – Used to store campaign link tracking information.

The Interact API provides control over many of these objects, allowing client application developers to create, change, or remove these objects in a programmatic way to accomplish a variety of marketing automation goals. A brief discussion of these objects is provided below. More information about these objects and their use in the application is available in the core Interact platform documentation.

Campaigns

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

- General properties – Name, Type (email or mobile), Description, Categorization, etc.
- Audience – List, Inclusion Filters, Exclusion Filters, Suppression data, etc.
- Message – From header, Reply-to header, Subject header, HTML/Text message documents
- Settings – Tracking options, Auto-close behavior, default variables, etc.

A campaign can be launched in bulk immediately or scheduled for launch. Messages from a campaign can also be triggered on demand by Form Handler Rules or Program Rules.

Forms

Form objects provide functionality for hosting web forms and collecting/processing submitted data. Forms can be used as preference centers or 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.

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.

Lists and Related Objects

Lists are used to store audience member records (leads, prospects, customers, contacts, consumers, or visitors, depending on your terminology). Lists have a standard set of fields that include:

- Recipient ID (RIID) – Internal Responsys Interact-assigned identifier that allows for tracking behavior over time for individual recipients
- Email Address, Mobile Number, Postal Address – 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 are used 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.

There are three types of List-based objects.

- **List Filters** – User-defined segments that contain a subset of the members of a List. **Note:** List Filters can be used to include or exclude members from any given campaign launch.

- **List Segmentations** – Provide a means of understanding how a List breaks down in terms of a given set of segments. **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. **Note**: These records do not represent real members (prospects, customers, etc.).

Supplemental Tables and Related Objects

Supplemental Tables (Tables) contain data that is not oriented toward defining a list member identity, but rather is oriented toward supplementing a List with additional data that is related to the List via some related key field (call the "Data Extraction Key"). A given Table schema is totally user-defined and, as a result, Tables can be used for a wide variety of uses, ranging from message personalization and dynamic content to storing form responses and campaign events.

There are four distinct types of Table-based data sources: Tables, Filters on Tables, SQL views (on Tables and/or Lists), and Joins on Tables. When you are using Tables for extending a List to represent a multi-table relational marketing database (where a variety of queries or joins could be made on the Table), be very careful to create proper indexes for these Tables to prevent unwanted performance impacts associated with full table scans on Tables being queried or joined. See Interact Documentation or Support staff for more information on the need for indexes.

Link Tables

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

Documents

Document objects contain the creative content that can be used for Campaigns and Forms. There are two subtypes of the Document object: HTML and Text. For example, an email campaign usually consists of an HTML and Text document reference. The campaign handles the packaging 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 the Interact Campaign user interface.

Interact API – Getting Started

To use this developer guide, you should have a basic familiarity with software development, SOAP-based Web Services, and the Responsys Interact platform and user interface. Brief instructions for getting started with the Interact API in a Java or C# application are provided in this section.

General Steps

In general, any Interact API client application project will involve the following key steps.

1. Use the Interact Web Services API WSDL to generate supporting code for creating the SOAP calls on the Interact API. Your development environment or programming language should provide 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. These login calls return a session identifier that should be placed in the SOAP header of all subsequent calls to the Interact API to authenticate the client application.
3. A session cookie (JSESSIONID) is placed 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. These may include any of the following types of calls:
    a. Data API calls to create, modify or delete individual records
    b. Connect API calls to import or export data in bulk
    c. Campaign API calls to create or modify campaign definitions or launch campaigns
    d. 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, and all data source merge, retrieve, and delete calls). Example: Interact API calls for triggering of campaign messages and merging of records into a List are limited to 200 recipients or records per invocation. Depending on your client application, you may need to execute these calls in a loop to cycle through all the records needing to be processed during the given client session.

1. If your client application is inactive for longer than two hours, the session identifier becomes invalid and your client application must make a new Login call to start a new session.
2. 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 to how many concurrent sessions can be created for each Interact account.

Java Applications

1. Download the WSDL document. Responsys Support will provide the Interact API URLs to you when your account is enabled for Interact API access. Name the downloaded file ResponsysWS.wsdl and place it somewhere in your project directory.
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.

A simple example is shown below.

```
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

1. Download the WSDL document. Responsys Support will provide the Web Services API URLs to you when your account is enabled for Web Services API access. Name the downloaded file ResponsysWS.wsdl.
2. To generate the client-side code needed to support your client application's programmatic calls on the Responsys Web service, do either of the following.
    a. 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
    b. This generates a single C# file, called ResponsysWSService.cs. Copy the ResponsysWSService.cs to your project directory for use in your client application.
3. In your C# application, get a handle for the Web Service, and ensure the user session will be maintained. A simple example is 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. Important: Be sure that csc.exe is in your path, usually:

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

```
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 Note for Users of the Microsoft.NET WSDL: Small Change Needed for RecordData Element

The ResponsysWS.wsdl has an element called **RecordData** that contains an array of **Record** elements. The **Record** element contains an array of Strings.

Issue

The Microsoft.NET wsdl.exe has a defect that impacts the case of having an "array inside an array."

Because of this defect, the wsdl.exe generates the RecordData class as follows:

```
/// <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 string[][] recordsField;

    /// <remarks/>

    [System.Xml.Serialization.XmlElementAttribute("fieldNames",
IsNullable=true)]

    public string[] fieldNames {

        get {

            return this.fieldNamesField;

        }

        set {

            this.fieldNamesField = value;

        }

    }
```

```
    /// <remarks/>

    [System.Xml.Serialization.XmlArrayAttribute(IsNullable=true)]

    [System.Xml.Serialization.XmlArrayItemAttribute("fieldValues",
typeof(string))]

    public string[] records {

        get {

            return this.recordsField;

        }

        set {

            this.recordsField = value;

        }

    }

}
```

Observe that the recordsField is created as a two-dimensional string array when it should be an array of Record class. Also, when you check the ResponsysWSService.cs class you can observe that the Record class is not created at all.

Solution

The solution for this issue is to manually edit the ResponsysWSService.cs class and make the following changes.

1. Create a **Record** class
2. Change the **string[][] recordsField** in RecordData class to **Record[] recordsField**.

Here are the above 2 changes.

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;

            }

        }

    }
```

Changing the Two-dimensional String Array in RecordData Class to an Array of Record Objects

```
    /// <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;
            }
        }
    }
```

Session Management API Calls

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.

Note that 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.

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 result = 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.

The detailed steps for using this call are listed below:

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 result = 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.

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.

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 that 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

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	New folder name to be created

Response

Name	Type	Description
result	boolean	Success flag for creation of 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

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

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.

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

public RecipientResult [] = mergeListMembersRIID(InteractObject list, RecordData recordData, ListMergeRule mergeRule) throws ListFault,UnexpectedErrorFault;

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.

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

Field = 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

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 a "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

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

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
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

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

RecordData = 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.

TruncateTable

Syntax

boolean = service.truncateTable(string folderName, string tableName)

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 creation of folder.

Content Management API Calls

CreateDocument

Syntax

boolean result = service.createDocument(InteractObject document, string content, CharacterEncoding characterEncoding)

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.

Request Arguments

Name	Type	Description
document	InteractObject	Document object to create
content	string	Text content of document (including markup for HTML content).
characterEncoding	CharacterEncoding	Character set of document content.

Response

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

DeleteDocument

Syntax

boolean result = service.deleteDocument(InteractObject document)

Usage

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

Request Arguments

Name	Type	Description
document	InteractObject	Document object to delete

Response

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

SetDocumentContent

Syntax

boolean result = service.setDocumentContent(InteractObject document, string content)

Usage

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

Request Arguments

Name	Type	Description
document	InteractObject	Document object
content	string	Text content to set for existing document.

Response

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

SetDocumentImages

Syntax

boolean result = service.setDocumentImages(InteractObject document, ImageData[] imageData)

Usage

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

Request Arguments

Name	Type	Description
document	InteractObject	Document object
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.

GetDocumentContent

Syntax

ContentResult result = service.getDocumentContent(InteractObject document)

Usage

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

Request Arguments

Name	Type	Description
document	InteractObject	Document object

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[] result = service.getDocumentImages(InteractObject document)

Usage

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

Request Arguments

Name	Type	Description
document	InteractObject	Document object

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.

Campaign Management API Calls

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

proofLaunchOptions	ProofLaunchOptions	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`

launchPreferences	LaunchPreferences	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

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

proofLaunchOptions	ProofLaunchOptions	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`

launchPreferences	LaunchPreferences	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.

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 List member identifier should be provided in the Recipient object (recipientId, customerId, email address, or mobile number).

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
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.

Request Arguments

Name	Type	Description
campaign	InteractObject	Campaign name
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. At least one List member identifier should be provided in the Recipient object (recipientId, customerId, email address, or mobile number). 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 the following URL: 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.

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

CharacterEncoding

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

Type	Restricted List of Following Values

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

ContentFormat

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

Type	Restricted List of Following 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.
eventStringDataMapping	string	A mapping to a name property in the OptionalData.
eventNumberDataMapping	string	A mapping to a name property in the OptionalData
eventDateDataMapping	string	A mapping to a name property in the OptionalData
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	Restricted List of Following Values

string	`TEXT_FORMAT`
	`HTML_FORMAT`
	`MULTIPART_FORMAT`
	`NO_FORMAT`

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.

Property Name	Type	Description
name	string	Folder name

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	Restricted List of Values

string	`STR500`
	`STR4000`
	`INTEGER`
	`NUMBER`
	`TIMESTAMP`

ImageData

The *imageData* object represents an image file.

Property 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 progess 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)
matchColumnName3	string	Third 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.
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.

LoginResult

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

Property 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	Restricted list of the following 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

Type	Restricted List of Following 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	Restricted list of the following values

string	`RIID`
	`CUSTOMER_ID`
	`EMAIL_ADDRESS`
	`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.

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	Restricted List of Following Values

string	`NO_OPTOUT_BUTTON`
	`OPTOUT_SINGLE_CLICK`
	`OPTOUT_FORM`

UpdateOnMatch

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

Type	Restricted List of Values

string	`NO_UPDATE`
	`REPLACE_ALL`
	`REPLACE_IF_EXISTING_BLANK`
	`REPLACE_IF_NEW_BLAN`

About Responsys

Responsys enables companies to increase revenue and customer loyalty through successful email and cross-channel marketing.

Responsys helps marketing organizations maximize their results by enabling every customer interaction to be highly automated and individualized, and every process to be highly collaborative, efficient, and error-free. With its on-demand, software-as-a-service (SaaS) delivery model and proven, Cross-Channel Lifecycle Marketing approach, Responsys offers the highest ROI, the lowest total cost of ownership, and the fastest time-to-value of any marketing solution available today.

Founded in 1998, Responsys is headquartered in San Bruno, California and is trusted by world-class brands such as Avis Europe, Chico's, Continental Airlines, Deutsche Lufthansa, Lands' End, LEGO, Men's Wearhouse, PayPal, Salesforce.com, Sears Holdings Corporation, StubHub, and UnitedHealthcare.