Version 11 Release 0
May 31 2018

IBM Interact Administrator's Guide

Note
Before using this information and the product it supports, read the information in "Notices" on page 363.

This edition applies to version 11, release 0, modification 0 of IBM Interact and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Administer IBM Interact

When you administer Interact you configure and maintain users and roles, data sources, and optional product features. You also monitor and maintain the design and runtime environments. Product-specific application programming interfaces (APIs) are available for you to use.

Administering Interact consists of several tasks. These tasks can include, but are not limited to:

- Maintaining users and roles
- Maintaining data sources
- Configuring Interact optional offer serving features
- Monitoring and maintaining runtime environment performance

Before you start administering Interact, there are some key concepts about how Interact works that you can familiarize yourself with to make your tasks easier. The sections that follow describe the administrative tasks that are associated with Interact.

The second part of the administration guide describes the APIs available with Interact:

- Interact API
- ExternalCallout API
- Learning API

Interact key concepts

IBM® Interact is an interactive engine that targets personalized marketing offers to various audiences.

This section describes some of the key concepts you should understand before you work with Interact.

Audience levels

An audience level is a collection of identifiers that can be targeted by a campaign. You can define audience levels to target the correct set of audiences for your campaign.

For example, a set of campaigns can use the audience levels "Household," "Prospect," "Customer," and "Account." Each of these levels represents a certain view of the marketing data available for a campaign.

Audience levels are typically organized hierarchically. Using the examples above:

- Household is at the top of the hierarchy, and each household can contain multiple customers and one or more prospects.
- Customer is next in the hierarchy, and each customer can have multiple accounts.
- Account is at the bottom of the hierarchy.

Other, more complex examples of audience hierarchies exist in business-to-business environments, where audience levels can exist for businesses, companies, divisions, groups, individuals, accounts, and so on.

These audience levels can have different relationships with each other, for example one-to-one, many-to-one, or many-to-many. By defining audience levels, you allow these concepts to be represented within Campaign so that users can manage the relationships among these different audiences for targeting purposes. For example, although there might be multiple prospects per household, you might want to limit mailings to one prospect per household.

Design environment

Use the design environment to configure various Interact components and deploy them to the runtime environment.

The design environment is where you complete most of your Interact configuration. In the design environment, you define events, interaction points, smart segments, and treatment rules. After you configure these components, you deploy them to the runtime environment.

The design environment is installed with the Campaign web application.

Events

An event is an action that is taken by a visitor and that triggers an action in the runtime environment. Examples of an event can be: placing a visitor into a segment, presenting an offer, or logging data.

Events are first created in an interactive channel and then triggered by a call to the Interact API by using the `postEvent` method. An event can lead to one or more of the following actions that are defined in the Interact design environment:

- **Trigger re-segmentation**: The runtime environment runs all the interactive flowcharts for the current audience level that is associated with the interactive channel again, by using the current data in the visitor's session.

  When you design your interaction, unless you specify a specific flowchart, a resegmentation action runs all interactive flowcharts that are associated with this interactive channel with the current audience level again, and that any request for offers waits until all flowcharts are finished. Excessive resegmentation within a single visit can affect the performance of the touchpoint in a customer-visible way.

  Place the customer in new segments after significant new data is added to the runtime session object, such as new data from requests from the Interact API (such as changing the audience) or customer actions (such as adding new items to a wish list or shopping cart).
- **Log offer contact**: The runtime environment flags the recommended offers for the database service to log the offers to contact history.

  For web integrations, log the offer contact in the same call where you request offers to minimize the number of requests between the touchpoint and the runtime server.

  If the touchpoint does not return the treatment codes for the offers that Interact presented to the visitor, the runtime environment logs the last list of recommended offers.
- **Log offer acceptance**: The runtime environment flags the selected offer for the database service to log to response history.

- **Log offer rejection**: The runtime environment flags the selected offer for the database service to log to response history.
- **Trigger user expression**: An *expression action* is an action that you can define by using Interact macros, including functions, variables, and operators, including EXTERNALCALLOUT. You can assign the return value of the expression to any profile attribute.

  When you click the edit icon next to Trigger User Expression, the standard User Expression editing dialog is displayed, and you can use this dialog to specify the audience level, optional field name to which to assign the results, and the definition of the expression itself.
- **Trigger events**: You can use the Trigger Events action to enter an event name that you want to be triggered by this action. If you enter an event that is already defined, that event is triggered when this action is run. If the event name you enter does not exist, this action causes the creation of that event with the specified action.

You can also use events to trigger actions that are defined by the `postEvent` method, including logging data to a table, including data to learning, or triggering individual flowcharts.

Events can be organized into categories for your convenience in the design environment. Categories have no functional purpose in the runtime environment.

Interactive channels

Use interactive channels in Interact to coordinate all the objects, data, and server resources that are involved in interactive marketing.

An interactive channel is a representation in Campaign of a touchpoint where the method of the interface is an interactive dialog. This software representation is used to coordinate all of the objects, data, and server resources that are involved in interactive marketing.

An interactive channel is a tool that you use to define interaction points and events. You can also access reports for an interactive channel from the Analysis tab of that interactive channel.

Interactive channels also contain production runtime and staging server assignments. You can create several interactive channels to organize your events and interaction points if you have only one set of production runtime and staging servers, or to divide your events and interaction points by customer-facing system.

Interactive flowcharts

Use interactive flowcharts to divide your customers into segments and assign a profile to a segment.

An interactive flowchart is related to but slightly different from a Campaign batch flowchart. Interactive flowcharts perform the same major function as batch flowcharts: dividing your customers in to groups known as segments. For interactive flowcharts, however, the groups are smart segments. Interact uses these interactive flowcharts to assign a profile to a segment when a behavioral event or system event indicates that a visitor re-segmentation is needed.

Interactive flowcharts contain a subset of the batch flowchart processes, and a few interactive flowchart-specific processes.

Note: Interactive flowcharts can be created in a Campaign session only.

Interaction points

An interaction point is a place in your touchpoint where you want to present an offer.

Interaction points contain default filler content in cases where the runtime environment does not have other eligible content to present. Interaction points can be organized into zones.

Offers

An offer represents a single marketing message, which can be delivered in various ways.

In Campaign, you create offers that can be used in one or more campaigns.

Offers are reusable:

- In different campaigns
- At different points in time
- For different groups of people (cells)
- As different "versions" by varying the offer's parameterized fields

You assign offers to interaction points in the touchpoints that are presented to visitors.

Profiles

A profile is the set of customer data that is used by the runtime environment. This data can be a subset of the customer data available in your customer database, data that is collected in real time, or a combination of the two.

The customer data is used for the following purposes:

- To assign a customer to one or more smart segments in real-time interaction scenarios.

  You need a set of profile data for each audience level by which you want to segment. For example, if you are segmenting by location, you might include only the customer's postal code from all the address information you have.
- To personalize offers
- As attributes to track for learning

  For example, you can configure Interact to monitor the marital status of a visitor and how many visitors of each status accept a specific offer. The runtime environment can then use that information to refine offer selection.

This data is read-only for the runtime environment.

Runtime environment

The runtime environment connects to your touchpoint and performs interactions. The runtime environment can consist of one or many runtime servers that are connected to a touchpoint.

The runtime environment uses the information that is deployed from the design environment in combination with the Interact API to present offers to your touchpoint.

Runtime sessions

A runtime session exists on the runtime server for each visitor to your touchpoint. This session holds all the data for the visitor that the runtime environment uses to assign visitors to segments and recommend offers.

You create a runtime session when you use the `startSession` call.

Touchpoints

A touchpoint is an application or place where you can interact with a customer. A touchpoint can be a channel where the customer initiates the contact (an "inbound" interaction) or where you contact the customer (an "outbound" interaction).

Common examples are websites and call center applications. Using the Interact API, you can integrate Interact with your touchpoints to present offers to customers based on their action in the touchpoint. Touchpoints are also called client-facing systems (CFS).

Treatment rules

Treatment rules assign an offer to a smart segment. These assignments are further constrained by the custom-defined zone you associate with the offer in the treatment rule.

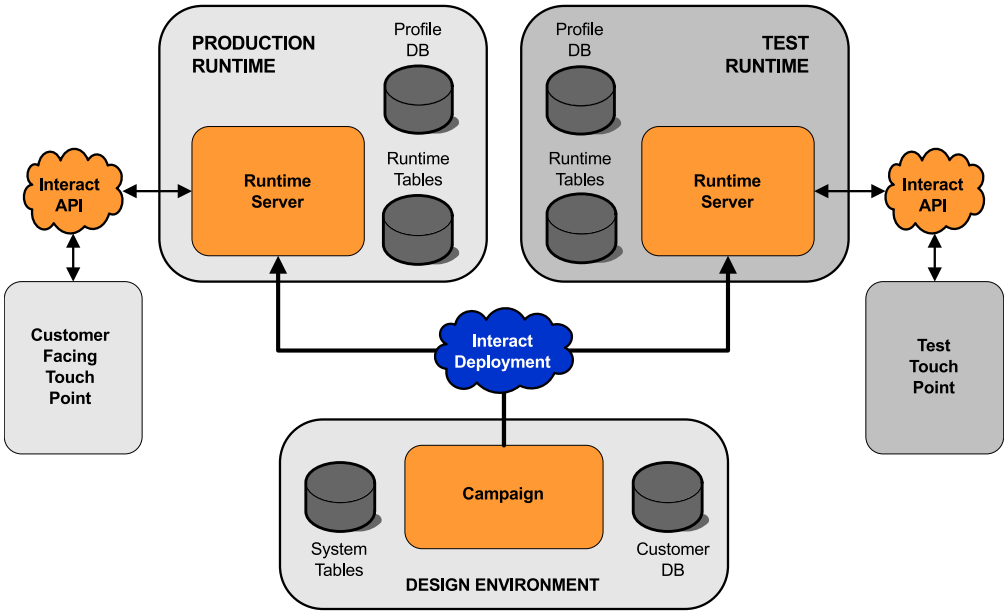
For example, you have one set of offers you assign a smart segment in the "login" zone, but a different set of offers for the same segment in the "after purchase" zone. Treatment rules are defined on an interaction strategy tab of a campaign.

Each treatment rule also has a marketing score. If a customer is assigned to more than one segment, and therefore more than one offer is applicable, the marketing scores help define which offer Interact suggests. Which offers the runtime environment suggests can be influenced by a learning module, an offer suppression list, and global and individual offer assignments.

Interact architecture

The Interact environment consists of at least two major components, design environment and the runtime environment. You may have optional testing runtime servers as well.

The following figure shows the high-level architecture overview.



The design environment is where you perform the majority of your Interact configuration. The design environment is installed with the Campaign web application and references the Campaign system tables and your customer databases. You use the design environment to define the interaction points and events you use with the API.

After you design and configure how you want the runtime environment to handle customer interactions, you deploy that data to either a staging server group for testing or a production runtime server group for real-time customer interaction.

The Interact API provides the connection between your touchpoint and the runtime server. You reference objects (interaction points and events) created in the design environment with the Interact API and use them to request information from the runtime server.

Interact network considerations

A production installation of Interact spans at least two machines. In a high-volume production environment, with several Interact runtime servers and distributed databases, your installation might span dozens of machines.

For best performance, there are several network topology requirements to consider.

- If your implementation of the Interact API starts and ends sessions in the same call, for example:

  ```
  executeBatch(startSession, getOffers, postEvent, endSession)
  ```

  you do not need to enable session persistence (sticky sessions) between the load balancer and the Interact runtime servers. You can configure the Interact runtime servers session management for local cache type.
- If your implementation of the Interact API uses multiple calls to start and end sessions, for example:

```
startSession
. . .
executeBatch(getOffers, postEvent)
. . .
endSession
```

and you are using a load balancer for your Interact runtime servers, you should enable some type of persistence for the load balancer (also known as sticky sessions). If that is not possible, or if you are not using a load balancer, configure the Interact servers session management for a distributed `cacheType`. If you are using a distributed cache, all the Interact runtime servers must be able to communicate via multicast. You may need to tune your network so that the communication between Interact servers using the same multicast IP address and port does not impede system performance. A load balancer with sticky sessions has better performance than using a distributed cache.

- Distributed caching among multiple server groups is not supported.
- Keep your runtime environment Interact servers, Marketing Platform, load balancers, and touchpoint) in the same geographic location for best performance. Design time and runtime can be in different geographic locations, but expect a slow deployment.
- Have a fast network connection (at least 1Gb) between the Interact production server group and its associated touchpoint.
- Design time requires http or https access to runtime to complete deployment tasks. Any firewalls or other network applications must be configured to allow deployment. You may need to extend the HTTP timeout lengths between the design environment and the runtime environments if you have large deployments.
- The contact and response history module requires access to the design time database (Campaign system tables) and access to the runtime database Interact runtime tables). You must configure your databases and network appropriately for this data transfer to occur.

In a testing or staging installation, you can install Interact design time and runtime on the same machine. This scenario is not recommended for a production environment.

Interact server ports and network security

Configure Interact t to secure your server ports.

Interact runtime ports

Some of these ports can be closed, or are not required by all Interact installations, depending on your configuration.

Interact application server port for HTTP
The default port where Interact requests are handled.

Interact application server port for HTTPS
The default SSL port where Interact requests are handled.

Interact systemTablesDataSource port
See the datasource's JDBC configuration in Marketing Platform.

Interact learningTablesDataSource port
See the datasource's JDBC configuration in Marketing Platform.

Interact contactAndResponseHistoryDataSource port
See the datasource's JDBC configuration in Marketing Platform.

Interact prodUserDataSource port
See the datasource's JDBC configuration in Marketing Platform.

Interact testRunDataSource port
See the datasource's JDBC configuration in Marketing Platform.

ETL communication port
Configure this port in **Interact | ETL | patternStateETL | communicationPort** in the configuration properties.

EHCache multicast port
Configure this port in **Interact | cacheManagement | Cache | Managers | EHCache | Parameter Data | multicastPort** in configuration properties when cache mode is distributed.

ExtremeScale catalog port
Configure this port in **Interact | Cache Managers | Extreme Scale | Parameter Data | catalogURLs** in configuration properties.

Interact JMX Monitoring port
Configure this port in **Interact | monitoring | port** under configuration properties or run `-Dinteract.jmx.monitoring.port=portNumber`.

Interact WebConnector port

This port is usually the same as the Interact server port, but it is modifiable in `jsconnector.xml`.

For the ports for any Interact integrated products, see the documentation for those products.

JMX monitoring is not required for typical Interact functionality. However, it is used for diagnostics and monitoring.

JMX port access can be disabled in the Interact configuration or limited to specific IP address through firewall configurations. This is recommended due to the JMX vulnerability recently found in the third party Apache Commons Library.

The JMX remoting functionality in Apache Geronimo 3.x before 3.0.1, as used in IBM WebSphere Application Server (WAS) Community Edition 3.0.0.3 and other products, does not properly implement the RMI classloader, which allows remote attackers to execute arbitrary code by using the JMX connector to send a crafted serialized object. See http://www-01.ibm.com/support/docview.wss?uid=swg21643282.

Interact design ports

Some of these ports can be closed, or are not required by all Interact installations, depending on your configuration.

Campaign application server port for HTTP
The default port where Interact requests are handled.

Campaign application server port for HTTPS
The default SSL port where Interact requests are handled.

Campaign listener port
The port that Campaign uses internally to accept connections from the web client.

Other Campaign design ports
See the Campaign documentation for more information on these ports.

Campaign JMX Connector port
Configure this port in **Campaign | monitoring | port** in configuration properties for contact response history monitorying only.

Campaign operational monitoring server port
Configure this port in **Campaign | monitoring | serverURL** in configuration properties.

Logging in to IBM Marketing Software

Use this procedure to log in to IBM Marketing Software.

You need the following.

- An intranet (network) connection to access your IBM Marketing Software server.
- A supported browser installed on your computer.
- User name and password to sign in to IBM Marketing Software.
- The URL to access IBM Marketing Software on your network.

The URL is:

`http://host.domain.com:port/unica`

where

host is the machine where Marketing Platform is installed.

domain.com is the domain in which the host machine resides.

port is the port number where the Marketing Platform application server is listening.

Note: The following procedure assumes that you are logging in with an account that has Admin access to Marketing Platform.

Access the IBM Marketing Software URL using your browser.

- If IBM Marketing Software is configured to integrate with Windows Active Directory or with a web access control platform, and you are logged in to that system, you see the default dashboard page. Your login is complete.
- If you see the login screen, log in using the default administrator credentials. In a single-partition environment, use `asm_admin` with `password` as the password. In a multi-partition environment, use `platform_admin` with `password` as the password.

  A prompt asks you to change the password. You can enter the existing password, but for good security you should choose a new one.
- If IBM Marketing Software is configured to use SSL, you may be prompted to accept a digital security certificate the first time you sign in. Click **Yes** to accept the certificate.

If your login is successful, IBM Marketing Software displays the default dashboard page.

With the default permissions assigned to Marketing Platform administrator accounts, you can administer user accounts and security using the options listed under the **Settings** menu. To perform the highest level administration tasks for IBM Marketing Software dashboards, you must log in as **platform_admin**.

Chapter 2. Configuring Users

Interact requires you to configure two sets of users, runtime environment users and design environment users.

- **Runtime users** are created in the Marketing Platform configured to work with the runtime servers.
- **Design time users** are Campaign users. Configure the security for the various members of your design team as for Campaign.

Configuring the runtime environment user

After you install Interact, you must configure at least one Interact user, the runtime environment user. Runtime users are created in Marketing Platform.

The runtime environment user provides access to the runtime tables. The runtime environment user is the user name and password you use to deploy interactive channels. The runtime server uses the web application server JDBC authentication for the database credentials. You do not have to add any runtime environment data sources to the runtime environment user.

An LDAP user and any Platform user can deploy an Interactive channel. The InteractAdminRole is not required to deploy the Interactive channel.

When you create runtime users:

- If you have separate Marketing Platform instances for each runtime server, you must create the same user and password on each. All runtime servers that belong to the same server group must share user credentials.
- If you use the database load utility, you must define the runtime tables as a data source with login credentials for the runtime environment in your configuration properties under `Interact > general > systemTablesDataSource`.
- If you enable security for JMX monitoring with the JMXMP protocol, you might need a separate user for JMX monitoring security.

See the Marketing Platform documentation for the steps to create the runtime users.

Configuring design environment users

Design environment users are Campaign users. You configure your design environment users in the same way you configure Campaign role permissions.

Some design environment users also require some Campaign permissions such as Custom Macros.

When you create design environment users:

- If you have any Campaign users who have permission to edit interactive flowcharts, give them access to the test run tables data source.
- If you installed and configured Interact, the following extra options are available for the default Global Policy and new policies.
-

Category	Permissions
Campaigns	• View Campaign Interaction Strategies - Ability to see but not edit interaction strategy tabs in a campaign. • Edit Campaign Interaction Strategies - Ability to make changes to interaction strategy tabs, including treatment rules. • Delete Campaign Interaction Strategies - Ability to remove interaction strategy tabs from campaigns. Deletion of an interaction strategy tab is restricted if the interaction strategy has been included in an interactive channel deployment. • Add Campaign Interaction Strategies - Ability to create new interaction strategy tabs in a campaign. • Initiate Campaign Interaction Strategy Deployments - Ability to mark an interaction strategy tab for deployment or undeployment.
Interactive Channels	• Deploy Interactive Channels - Ability to deploy an interactive channel to Interact runtime environments. • Edit Interactive Channels - Ability to make changes to the summary tab of interactive channels. • Delete Interactive Channels - Ability to remove interactive channels. Deletion of interactive channels is restricted if the interactive channel has been deployed. • View Interactive Channels - Ability to see but not edit interactive channels. • Add Interactive Channels - Ability to create new interactive channels. • View Interactive Channel Reports - Ability to see the analysis tab of the interactive channel. • Add Interactive Channel Child Objects - Ability to add interaction points, zones, events, and categories.
Sessions	• View Interactive Flowcharts - Ability to see an interactive flowchart in a session. • Add Interactive Flowcharts - Ability to create new interactive flowcharts in a session. • Edit Interactive Flowcharts - Ability to make changes to interactive flowcharts. • Delete Interactive Flowcharts - Ability to remove interactive flowcharts. Deletion of interactive flowcharts is restricted if the interactive channel to which this interactive flowchart is assigned has been deployed. • Copy Interactive Flowcharts - Ability to copy interactive flowcharts. • Test Run Interactive Flowcharts - Ability to initiate a test run of an interactive flowchart. • Review Interactive Flowcharts - Ability to see an interactive flowchart and open processes to view settings, but unable to make changes. • Deploy Interactive Flowcharts - Ability to mark an interactive flowcharts for deployment or undeployment.

Example design environment permissions

This example lists the permissions that are granted to two different roles, one for users who create interactive flowcharts, and one for users who define interaction strategies.

Interactive flowchart role

This table shows the permissions that are given to the interactive flowchart role:

Category	Permission
Custom Macro	The user role has these permissions: • Add Custom Macros • Edit Custom Macros • Use Custom Macros
Derived Field	The user role has these permissions: • Add Derived Fields • Edit Derived Fields • Use Derived Fields
Flowchart Template	The user role has these permissions: • Paste Templates
Segment Template	The user role has these permissions: • Add Segments • Edit Segments
Session	The user role has these permissions: • View Session Summary • View Interactive Flowcharts • Add Interactive Flowcharts • Edit Interactive Flowcharts • Copy Interactive Flowcharts • Test Run Interactive Flowcharts • Deploy Interactive Flowcharts

Interaction strategy role

This table shows the permissions that are given to the interaction strategy role:

Category	Permission
Campaign	The user role has these permissions: • View Campaign Summary • Manage Campaign Target Cells • View Campaign Interaction Strategies • Edit Campaign Interaction Strategies • Add Campaign Interaction Strategies • Initiate Campaign Interaction Strategy Deployments
Offer	The user role has these permissions: • View Offer Summary

Category	Permission
Segment Template	The user role has these permissions: • View Segment Summary
Session	The user role has these permissions: • Review Interactive Flowcharts

Chapter 3. Managing Interact data sources

Interact requires several data sources to function properly. Some data sources contain the information Interact requires to function, other data sources contain your data.

The following sections describe the Interact data sources including information you need to configure them correctly, and some suggestions for maintaining them.

Interact data sources

Interact requires several sets of data to function. The sets of data are stored and retrieved from data sources, and the data sources you set up depend on the Interact features you are enabling.

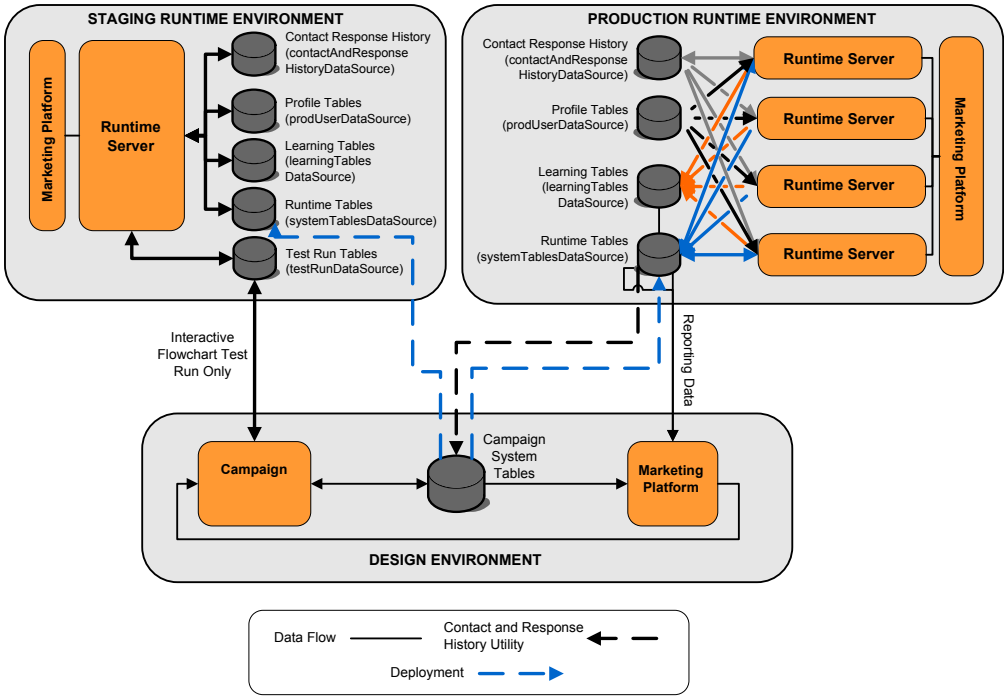
- **Campaign system tables**. Beyond all the data for Campaign, the Campaign system tables contain data for Interact components that you create in the design environment, such as treatment rules and interactive channels. The design environment and the Campaign system tables use the same physical database and schema.
- **Runtime tables**(`systemTablesDataSource`). This data source contains the deployment data from the design environment, staging tables for contact and response history, and runtime statistics.
- **Profile tables** (`prodUserDataSource`). This data source contains any customer data, beyond information that is gathered in real time, that is required by interactive flowcharts to properly place visitors into smart segments. If you are relying entirely on real-time data, you do not need profile tables. If you are using profile tables, you must have at least one profile table per audience level that is used by the interactive channel.

  The profile tables can also contain the tables that are used for augmenting offer serving, including tables for offer suppression, score override, and global and individual offer assignment.
- **Test run tables** (`testRunDataSource`). This data source contains a sample of all data that is required by interactive flowcharts to place visitors into smart segments, including data that mimics what is gathered in real time during an interaction. These tables are required for the server group that is designated as the test run server group for the design environment only.
- **Learning tables** (`learningTablesDataSource`). This data source contains all data that is gathered by the built-in learning utility. These tables can include a table that defines dynamic attributes. If you are not using learning or are using an external learning utility that you create, you do not need learning tables.
- **Contact and response history for cross-session response** (`contactAndResponseHistoryDataSource`). This data source contains either the Campaign contact history tables or a copy of them. If you are not using the cross-session response feature, you do not need to configure these contact history tables.

Databases and the applications

The data sources that you create for use by Interact might also be used to exchange or share data with other IBM Marketing Software applications.

The following diagram shows Interact data sources and how they relate to IBM Marketing Software applications.



- Both Campaign and the test run server group access the test run tables.
- The test run tables are used for interactive flowchart test runs only.
- When you are using a runtime server to test a deployment, including the Interact API, the runtime server uses the profile tables for data.
- If you configure the contact and response history module, the module uses a background Extract, Transform, Load (ETL) process to move data from the runtime staging tables to the Campaign contact and response history tables.
- The reporting function queries data from the learning tables, runtime tables, and the Campaign system tables to display reports in Campaign.

You should configure the testing runtime environments to use a different set of tables than your production runtime environments. With separate tables between staging and production, you can keep your testing results separate from your actual results. Note that the contact and response history module always inserts data into the actual Campaign contact and response history tables (Campaign has no testing contact and response history tables). If you have separate learning tables for the testing runtime environment, and you want to see the results in reports, you need a separate instance of IBM Cognos® BI running the learning reports for the testing environment.

Campaign system tables

When you install the Interact design environment, you also create new, Interact-specific tables in the Campaign system tables. The tables that you create depend on the Interact features you are enabling.

If you enable the contact and response history module, the module copies contact and response history from staging tables in the runtime tables to the contact and response history tables in the Campaign system tables. The default tables are

UA_ContactHistory, UA_DtlContactHist, and UA_ResponseHistory, but the contact and response history module uses whichever tables are mapped in Campaign for the contact and response history tables.

If you use the global offers tables and the score override tables to assign offers, you may need to populate the UACI_ICBatchOffers table in the Campaign system tables if you are using offers not contained in the treatment rules for the Interactive Channel.

Runtime tables

If you have more than one audience level, you must create staging tables for the contact and response history data for each audience level.

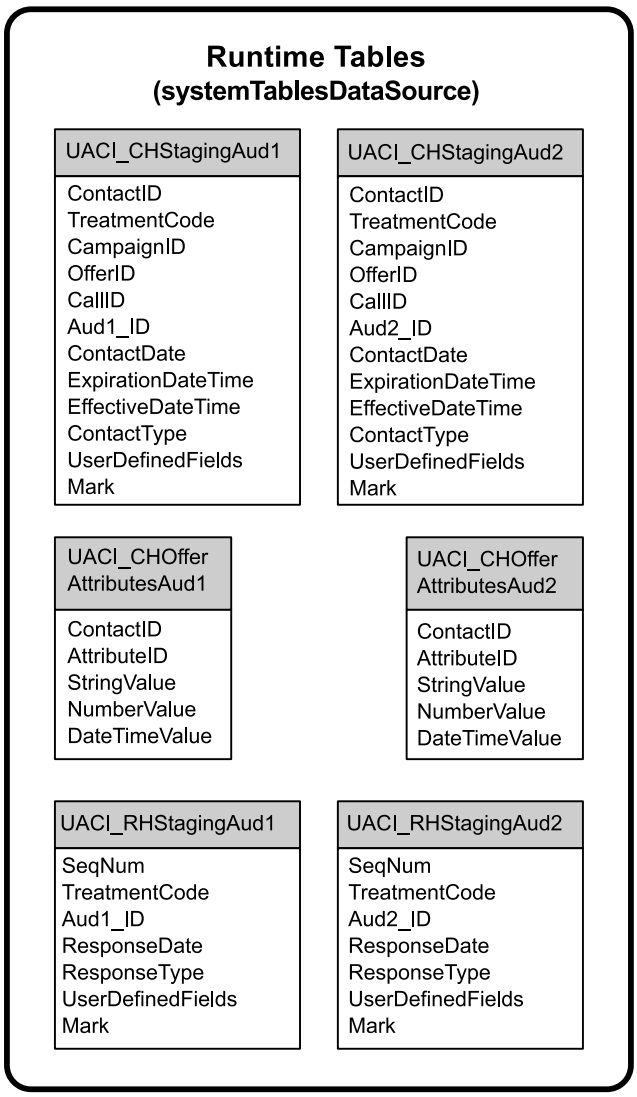
The SQL scripts create the following tables for the default audience level:

- UACI_CHStaging
- UACI_CHOfferAttrib
- UACI_RHStaging

You must create copies of these three tables for each of your audience levels in the runtime tables.

If your Campaign contact and response history tables have user defined fields, you must create the same field names and types in the UACI_CHStaging and UACI_RHStaging tables. You can populate these fields during runtime by creating name-value pairs of the same name in session data. For example, your contact and response history tables contain the field catalogID. You must add the catalogID field to both the UACI_CHStaging and UACI_RHStaging tables. Later, the Interact API populates this field by defining an event parameter as a name-value pair named catalogID. Session data can be supplied by the profile table, temporal data, learning, or the Interact API.

The following diagram shows sample tables for the audiences Aud1 and Aud2. This diagram does not include all tables in the runtime database.



All fields in the tables are required. You can modify the `CustomerID` and the `UserDefinedFields` to match your Campaign contact and response history tables.

Test run tables

The test run tables are used for test runs of interactive flowcharts only. Test runs of interactive flowcharts should test your segmentation logic. You only need to configure one test run database for your Interact installation. The test run tables do not need to be in a stand-alone database. You could, for example, use your customer data tables for Campaign.

The database user associated with the test run tables must have CREATE privileges to add the test run result tables.

The test run database must contain all tables mapped in the interactive channel.

These tables should contain data to run scenarios you want to test in your interactive flowcharts. For example, if your interactive flowcharts have logic to sort people into segments based on the choice selected in a voice mail system, you should have at least one row for every possible selection. If you are creating an

interaction that works with a form on your web site, you should include rows representing missing or malformed data, for example, use name@domaincom for the value of an email address.

Each test run table must contain at least a list of IDs for the appropriate audience level, and a column representing the real time data you expect to use. Since test runs do not have access to real time data, you must supply sample data for every piece of expected real time data. For example, if you want to use data you can collect in real time, such as the name of the last web page visited, stored in the attribute lastPageVisited, or the number of items in a shopping cart, stored in the attribute shoppingCartItemCount, you must create columns with the same names, and populate the columns with sample data. This allows you to test run the branches of your flowchart logic that are behavioral or contextual in nature.

Test runs of interactive flowcharts are not optimized for working with large sets of data. You can limit the number of rows used for the test run in the Interaction process. However, this always results in the first set of rows being selected. To ensure that different sets of rows are selected, use different views of the test run tables.

To test the throughput performance of interactive flowcharts in runtime, you must create a test runtime environment, including a profile table for the testing environment.

In practice, you may need three sets of tables for testing, a test run table for test runs of interactive flowcharts, test profile tables for the testing server group, and a set of production profile tables.

Overriding the default data types used for dynamically created tables

The Interact runtime environment dynamically creates tables under two scenarios: during a test run of a flowchart and during the running of a Snapshot process that writes to a table that doesn't already exist. To create these tables, Interact relies on hardcoded data types for each supported database type.

You can override the default data types by creating a table of alternate data types, named uaci_column_types, in the testRunDataSource or prodUserDataSource. This additional table allows Interact to accommodate rare cases that aren't covered by the hardcoded data types.

When the uaci_column_types table is defined, Interact uses the metadata for the columns as the data types to be used for any table generation. If the uaci_column_types table is not defined, or if there are any exceptions encountered while trying to read the table, the default data types are used.

At startup, the runtime system first checks the testRunDataSource for the uaci_column_types table. If the uaci_column_types table does not exist in the testRunDataSource, or if the prodUserDataSource is of a different database type, Interact then checks the prodUserDataSource for the table.

Overriding the default data types

Use this procedure to override the default data types for dynamically created tables.

You must restart the runtime server whenever you change the uaci_column_types table. Plan to make your changes so that restarting the server has minimal affect on operations.

1. Create a table in the TestRunDataSource or ProdUserDataSource with the following properties:

   Table Name: uaci_column_types

   Column Names:

   - uaci_float
   - uaci_number
   - uaci_datetime
   - uaci_string

   Use the appropriate data type supported by your database to define each column.
2. Restart the runtime server to allow Interact to recognize the new table.

Default data types for dynamically created tables

For each supported database that the Interact runtime system uses, there are hardcoded data types used by default for float, number, date/time, and string columns.

Table 1. Default data types for dynamically-created tables

Database	Default data types
DB2®	• float • bigint • timestamp • varchar
Informix®	• float • int8 • DATETIME YEAR TO FRACTION • char2
Oracle	• float • number(19) • timestamp • varchar2
SQL Server	• float • bigint • datetime • nvarchar

Profile database

The contents of the profile database depend entirely on the data you need for configuring your interactive flowcharts and Interact API. Interact requires or recommends that each database contain certain tables or data.

The profile database must contain the following:

- All tables mapped in the interactive channel.

These tables must contain all the data required for running your interactive flowcharts in production. These tables should be flattened, streamlined, and properly indexed. As there is a performance cost to access dimensional data, you should use a denormalized schema whenever possible. At a minimum, you should index the profile table on the audience level ID fields. If there are other fields retrieved from dimensional tables, these should be indexed appropriately to reduce database fetch time. The Audience IDs for the profile tables must match the Audience IDs defined in Campaign.

- If you set the `enableScoreOverrideLookup` configuration property to true, you must include a score override table for at least one audience level. You define the score override table names with the `scoreOverrideTable` property.

  The score override table can contain individual customer-to-offer pairings. You can create a sample score override table, `UACI_ScoreOverride` by running the `aci_usrtab` SQL script against your profile database. You should also index this table on the Audience ID column.

  If you set the `enableScoreOverrideLookup` property to false, you do not need to include a score override table.
- If you set the `enableDefaultOfferLookup` configuration property to true, you must include the global offers table (`UACI_DefaultOffers`). You can create the global offers table by running the `aci_usrtab` SQL script against your profile database.

  The global offers table can contain audience-to-offer pairings.
- If you set the `enableOfferSuppressionLookup` property to true, you must include an offer suppression table for at least one audience level. You define the offer suppression table names with the `offerSuppressionTable` property.

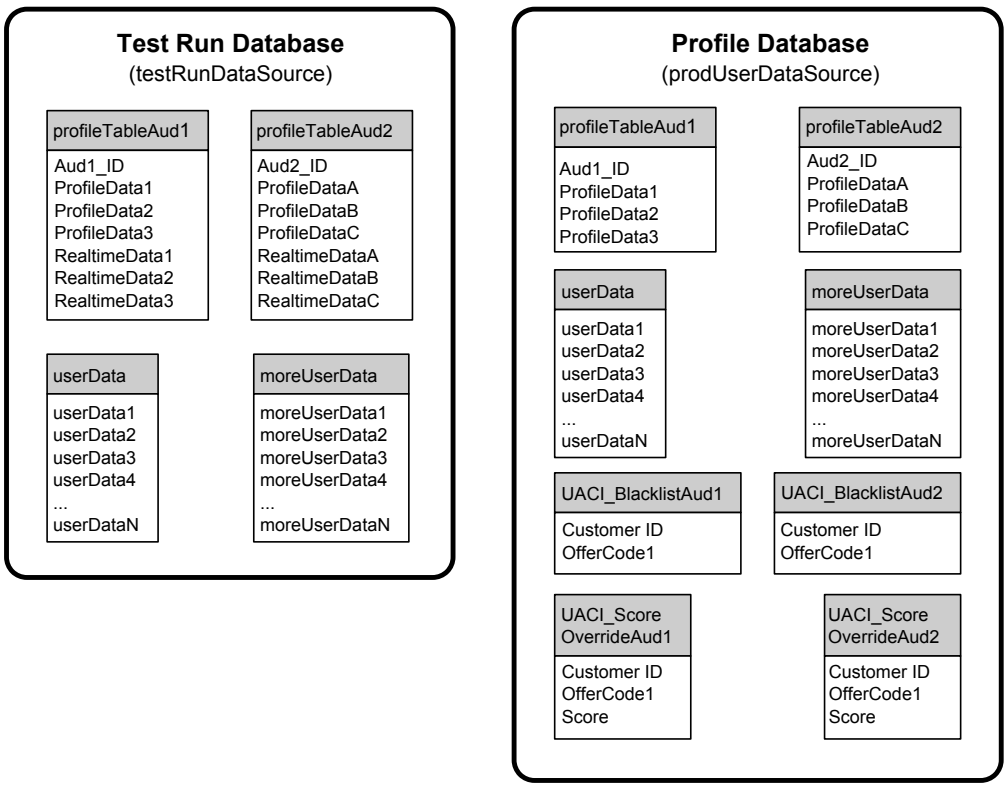
  The offer suppression table can contain a row for each offer suppressed for an audience member, although an entry is not required for all audience members. You can create a sample offer suppression table, `UACI_BlackList` by running the `aci_usrtab` SQL script against your profile database.

  If you set the `enableOfferSuppressionLookup` property to false, you do not need to include an offer suppression table.

A large amount of data in any of these tables may impede performance. For best results, put appropriate indexes on the audience level columns for tables used at runtime that have large amounts of data.

All configuration properties referenced above are in the **Interact > profile** or the **Interact > profile > Audience Levels > *AudienceLevel*** category. The `aci_usrtab` SQL script is located in the `ddl` directory in your runtime environment installation directory.

The following diagram shows example tables for the test run and profile databases for the audience levels Aud1 and Aud2.



Learning tables

If you are using Interact built-in learning, you must configure the learning tables. These tables contain all the data the built-in learning feature learns on.

If you are using dynamic learning attributes, you must populate the `UACI_AttributeList` table.

Learning involves writing to intermediate staging tables and aggregating information from staging tables to learning tables. The `insertRawStatsIntervalInMinutes` and `aggregateStatsIntervalInMinutes` configuration properties in the `Interact > offerserving > Built-in Learning Config` category determine how often the learning tables get populated.

The `insertRawStatsIntervalInMinutes` attribute determines how often the accept and contact information for each customer and offer is moved from memory to the staging tables, `UACI_OfferStatsTX` and `UACI_OfferTxAll`. The information stored in the staging tables is aggregated and moved to `UACI_OfferStats` and `UACI_OfferStatsAll` tables at regular intervals determined by the `aggregateStatsIntervalInMinutes` configuration property.

Interact built-in learning uses this data to calculate final scores for offers.

Contact history for cross-session response tracking

If you enable the cross-session response feature, the runtime environment needs read-only access to the Campaign contact history tables. You can configure the runtime environment to view the Campaign system tables, or you can create a copy of the Campaign contact history tables. If you create a copy of the tables, you

must manage the process of keeping the copy up to date. The contact and response history module will not update the copy of the contact history tables.

You must run the `aci_crhtab` SQL script against these contact history tables to add tables required for the cross-session response tracking feature.

Running database scripts to enable features

To use the optional features that are available in , run database scripts against the database to create tables or update existing tables.

Your installation, both the design time environment and runtime environment, includes feature **ddl** scripts. The **ddl** scripts add required columns to your tables.

To enable any of the optional features, run the appropriate script against the database or table that is indicated.

`dbType` is the database type, such as `sqlsvr` for Microsoft SQL Server, `ora` for Oracle, or `db2` for IBM DB2.

Use the following table to run database scripts against the database to create tables or update existing tables:

Table 2. Database scripts

Feature Name	Feature Script	Run Against	Change
Global offers, offer suppression, and score override	**aci_usrtab_***dbType***.sql** in *Interact_Home*\ddl\acifeatures\ (Runtime environment installation directory)	Your profile database (userProdDataSource)	Creates the UACI_DefaultOffers, UACI_BlackList, and UACI_ScoreOverride tables.
Scoring	**aci_scoringfeature_***dbType***.sql** in *Interact_Home*\ddl\acifeatures\ (Runtime environment installation directory)	Score override tables in your profile database (userProdDataSource)	Adds the LikelihoodScore and AdjExploreScore columns.
Learning	**aci_lrnfeature_***dbType***.sql** in *Interact_Home*\interactDT\ddl\acifeatures\ (Design time environment installation directory)	Campaign database that contains your contact history tables	Adds the columns RTSelectionMethod, RTLearningMode, and RTLearningModelID to the UA_DtlContactHist table. Also adds the columns RTLearningMode and RTLearningModelID to the UA_ResponseHistory table. This script is also required by the reporting features provided by the optional Reports Pack.

About contact and response history tracking

You can configure runtime environment to record contact and response history in the Campaign contact and response history tables. The runtime servers store contact and response history in staging tables. The contact and response history module copies this data from the staging tables to Campaign contact and response history tables.

The contact and response history module functions only if you set the Campaign > partitions > partition1 > Interact > interactInstalled and contactAndResponseHistTracking > isEnabled properties on the Configuration page for the design environment to yes.

If you are using the cross-session response tracking module, the contact and response history module is a separate entity.

Contact and response types

You can record one contact type and two response types with Interact. You can also record more custom response types with the postEvent method.

contactAndResponseHistTracking table properties

This table lists the properties that are found in the contactAndResponseHistTracking category:

Event	Contact/response type	Configuration Property
Log Offer Contact	Contact	contacted
Log Offer Acceptance	Response	accept
Log Offer Rejection	Response	reject

UA_UsrResponseType table properties

Ensure the CountsAsResponse column of the UA_UsrResponseType table in the Campaign system tables is configured properly. All of these response types must exist in the UA_UsrResponseType table.

To be a valid entry in the UA_UsrResponseType table, you must define a value for all the columns in the table, including CountsAsResponse. Valid values for CountsAsResponse are:

- 0 - no response
- 1 - a response
- 2 - a reject
-

These responses are used for reporting.

Additional Contact types

In Interact, you can use the postEvent method in the Interact API to trigger a contact event. You can also augment the system to allow the postEvent call to record additional custom contact types. All of these contact types must exist in the UA_ContactStatus table in the Campaign system tables. Using specific event parameters to the postEvent method, you can record additional contact types and define whether it is true contact or not. To log additional contact types, you must add the following event parameters:

UACIContactStatusCode - a string representing a contact type code. The value must be a valid entry in the UA_ContactStatus table. To be a valid entry in the UA_ContactStatus you must define all of the columns in the table, including

CountsAsContact. Valid values for CountsAsContact are 0 and 1. 0 indicates not as successful contact, 1 indicates as a successful contact.

Additional response types

In Interact, you can use the postEvent method in the Interact API to trigger an event which logs an "accept" or "reject" action for an offer. You can also augment the system to allow the `postEvent` call to record additional response types, such as Explore, Consider, Commit, or Fulfill.

All of these response types must exist in the `UA_UsrResponseType` table in the Campaign system tables. Using specific event parameters to the `postEvent` method, you can record additional response types and define whether an accept should be included in learning.Also its suggested not to post multiple responses (Accept/Reject) for single contact , as it may result in incorrect learning scores.

To log additional response types, you must add the following event parameters:

- **UACIResponseTypeCode** - a string representing a response type code. The value must be a valid entry in the `UA_UsrResponseType` table.

  To be a valid entry in the `UA_UsrResponseType` you must define all of the columns in the table, including `CountsAsResponse`. Valid values for `CountsAsResponse` are 0, 1, or 2. 0 indicates no response, 1 indicates a response, and 2 indicates a reject. These responses are used for reporting.
- **UACILogToLearning** -

  The parameter 'UACILogToLearning' is deprecated in version 11.0. Instead, the actual values defined in ' UA_ContactStatus' and 'UA_UsrResponseType' tables from Campaign database along with the values defined in the
  'Affinium | interact | services | contactHist | contactStatusCodes' and
  'Affinium | interact | services | responseHist | responseTypeCodes ' *parameters would be considered by the Interact system.*
  '

  If you pass 'UACILogToLearning= 1' in a postevent call, then the configured action associated to the response type/contact status will be ignored and this event is always treated as a true response/contact.

You may want to create several events with the Log Offer Acceptance action, one for every response type you want to log, or a single event with the Log Offer Acceptance action you use for every `postEvent` call you use to log separate response types.

For example, create an event with the Log Offer Acceptance action for each type of response. You define the following custom responses in the `UA_UsrResponseType` table [as Name (code)]: Explore (EXP), Consider (CON), and Commit (CMT). You then create three events and name them LogAccept_Explore, LogAccept_Consider, and LogAccept_Commit. All three events are exactly the same (have the Log Offer Acceptance action), but the names are different so that the person working with the API can distinguish between them.

Or, you could create a single event with the Log Offer Acceptance action that you use for all custom response types. For example, name it LogCustomResponse.

When working with the API, there is no functional difference between the events, but the naming conventions may make the code clearer. Also, if you give each custom response a separate name, the Channel Event Activity Summary report displays more accurate information.

First, set up all the name-value pairs

```
//Define name value pairs for the UACIResponseTypeCode
// Response type Explore
NameValuePair responseTypeEXP = new NameValuePairImpl();
responseTypeEXP.setName("UACIResponseTypeCode");
responseTypeEXP.setValueAsString("EXP");
responseTypeEXP.setValueDataType(NameValuePair.DATA_TYPE_STRING);

// Response type Consider
NameValuePair responseTypeCON = new NameValuePairImpl();
responseTypeCON.setName("UACIResponseTypeCode");
responseTypeCON.setValueAsString("CON");
responseTypeCON.setValueDataType(NameValuePair.DATA_TYPE_STRING);

// Response type Commit
NameValuePair responseTypeCMT = new NameValuePairImpl();
responseTypeCMT.setName("UACIResponseTypeCode");
responseTypeCMT.setValueAsString("CMT");
responseTypeCMT.setValueDataType(NameValuePair.DATA_TYPE_STRING);

//Define name value pairs for UACILOGTOLEARNING
//Does not log to learning
NameValuePair noLogToLearning = new NameValuePairImpl();
noLogToLearning.setName("UACILOGTOLEARNING");
noLogToLearning.setValueAsString("0");
noLogToLearning.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

//Logs to learning
NameValuePair LogToLearning = new NameValuePairImpl();
LogToLearning.setName("UACILogToLearning");
LogToLearning.setValueAsString("1");
LogToLearning.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
```

This first example shows using the individual events.

```
//EXAMPLE 1: This set of postEvent calls use the individually named events
//PostEvent with an Explore response
NameValuePair[] postEventParameters = { responseTypeEXP, noLogToLearning };
response = api.postEvent(sessionId, LogAccept_Explore, postEventParameters);

//PostEvent with a Consider response
NameValuePair[] postEventParameters = { responseTypeCON, noLogToLearning };
response = api.postEvent(sessionId, LogAccept_Consider, postEventParameters);

//PostEvent with a Commit response
NameValuePair[] postEventParameters = { responseTypeCOM, LogToLearning };
response = api.postEvent(sessionId, LogAccept_Commit, postEventParameters);
```

This second example shows using just one event.

```
//EXAMPLE 2: This set of postEvent calls use the single event
//PostEvent with an Explore response
NameValuePair[] postEventParameters = { responseTypeEXP, noLogToLearning };
response = api.postEvent(sessionId, LogCustomResponse, postEventParameters);

//PostEvent with a Consider response
NameValuePair[] postEventParameters = { responseTypeCON, noLogToLearning };
response = api.postEvent(sessionId, LogCustomResponse, postEventParameters);

//PostEvent with a Commit response
NameValuePair[] postEventParameters = { responseTypeCOM, LogToLearning };
response = api.postEvent(sessionId, LogCustomResponse, postEventParameters);
```

Both examples perform exactly the same actions, however, one version may be easier to read than the other.

Runtime environment staging tables to Campaign history tables mapping

The Interact contact history staging tables map to Campaign history tables. You must have one of the runtime environment staging tables for each audience level.

UACI_CHStaging contact history staging table mapping

This table shows how the `UACI_CHStaging` runtime environment staging table maps to the Campaign contact history table. The table names that are shown are the sample tables that are created for the default audience in the runtime tables and the Campaign system tables.

Table 3. Contact History

UACI_CHStaging **Interact contact history staging table column name**	**Campaign contact history table**	**Table column name**
ContactID	N/A	N/A
TreatmentCode	UA_Treatment	TreatmentCode
CampaignID	UA_Treatment	CampaignID
OfferID	UA_Treatment	OfferID
CellID	UA_Treatment	CellID
CustomerID	UA_DtlContactHist	CustomerID
ContactDate	UA_DtlContactHist	ContactDateTime
ExpirationDateTime	UA_Treatment	ExpirationDateTime
EffectiveDateTime	UA_Treatment	EffectiveDateTime
ContactType	UA_DtlContactHist	ContactStatusID
ContactStatusCode	UA_DtlContactHist	ContactStatusId
UserDefinedFields	UA_DtlContactHist	UserDefinedFields

`ContactID` is a key to join the `UACI_CHOfferAtrib` table with the `UACI_CHStaging` table. The `userDefinedFields` column can contain any data that you choose.

UACI_CHOfferAttrib contact history staging table mapping

This table shows how the `UACI_CHOfferAttrib` runtime environment staging table maps to the Campaign contact history table. The table names that are shown are the sample tables that are created for the default audience in the runtime tables and the Campaign system tables.

Table 4. Offer attributes

UACI_CHOfferAttrib **Interact contact history staging table column name**	**Campaign contact history table**	**Table column name**
ContactID	N/A	N/A
AttributeID	UA_OfferHistAttrib	AttributeID
StringValue	UA_OfferHistAttrib	StringValue
NumberValue	UA_OfferHistAttrib	NumberValue

Table 4. Offer attributes (continued)

UACI_CHOfferAttrib **Interact contact history staging table column name**	**Campaign contact history table**	**Table column name**
DateTimeValue	UA_OfferHistAttrib	DateTimeValue

UACI_RHStaging contact response history staging table mapping

This table shows how the `UACI_RHStaging` runtime environment staging table maps to the Campaign response history table. The table names that are shown are the sample tables that are created for the default audience in the runtime tables and the Campaign system tables.

Table 5. Response history

UACI_RHStaging **Interact response history staging table column name**	**Campaign response history table**	**Table column name**
SeqNum	N/A	N/A
TreatmentCode	UA_ResponseHistory	TreatmentInstID
CustomerID	UA_ResponseHistory	CustomerID
ResponseDate	UA_ResponseHistory	ResponseDateTime
ResponseType	UA_ResponseHistory	ResponseTypeID
UserDefinedFields	UA_ResponseHistory	UserDefinedFields

`SeqNum` is a key that is used by the contact and response history module to identify data, but is not recorded in the Campaign response tables. The `userDefinedFields` column can contain any data that you choose.

Additional columns in staging tables

If you add columns to the staging tables, the contact and response history module writes them to the `UA_DtlContactHist` or `UA_ResponseHistory` tables in columns of the same name.

For example, if you add the column `linkFrom` to your `UACI_CHStaging` table, the contact and response history module copies that data to the `linkFrom` column in the `UA_DtlContactHist` table.

Additional columns in Campaign contact and response history tables

If you have additional columns in your Campaign contact and response history tables, add matching columns to the staging tables before you run the contact and response history module.

You populate extra columns in the staging tables by creating columns with the same names as your name-value pairs in your runtime session data.

For example, you create name-value pairs `NumberItemsInWishList` and `NumberItemsInShoppingCart` and add them to your `UACI_RHStaging` table. When a

Log Offer Acceptance or Log Offer Rejection event occurs, the runtime environment populates those fields. The runtime environment populates the UACI_CHStaging table when a Log Offer Contact event occurs.

Use tables to include a score for an offer

You can use the user-defined fields to include the score that is used to present an offer. Add a column that is named FinalScore to both the UACI_CHStaging table in the runtime tables and the UA_DtlContactHist table in the Campaign system tables. Interact automatically populates the FinalScore column with the final score used for the offer if you are using built-in learning.

If you are building a customized learning module, you can use the setActualValueUsed method of the ITreatment interface and the logEvent method of the ILearning interface.

If you are not using learning, add a column that is named Score to both the UACI_CHStaging table in the runtime tables and the UA_DtlContactHist table in the Campaign system tables. Interact automatically populates the Score column with the score used for the offer.

Create new history tables in the Campaign and staging tables in the Interact

If you are using an audience level other than the Customer, then you will have to create new history tables in the Campaign, and new staging tables in the Interact.

For example, the below sample script is used in the IBM DB2 design time database to create history tables in the Campaign for an audience level of type Account.

```
DROP TABLE ACCT_UA_ResponseHistory;
DROP TABLE ACCT_UA_DtlContactHist;
DROP TABLE ACCT_UA_ContactHistory;
CREATE TABLE ACCT_UA_ResponseHistory (
        AccountID           varchar(30) NOT NULL,
        TreatmentInstID      bigint NOT NULL,
        ResponsePackID       bigint NOT NULL,
        ResponseDateTime     timestamp NOT NULL,
        WithinDateRangeFlg   int,
        OrigContactedFlg     int,
        BestAttrib           int,
        FractionalAttrib     float,
        DirectResponse       int,
        CustomAttrib         float,
        ResponseTypeID       bigint,
        DateID               bigint,
        TimeID               bigint,
        UserDefinedFields    char(18),
        CONSTRAINT ACCT_cRespHistory_PK
               PRIMARY KEY (AccountID, TreatmentInstID,
               ResponsePackID )
 );
 CREATE TABLE ACCT_UA_ContactHistory (
        AccountID           varchar(30) NOT NULL,
        CellID               bigint NOT NULL,
        PackageID            bigint NOT NULL,
        ContactDateTime      timestamp,
        UpdateDateTime       timestamp,
        ContactStatusID      bigint,
        DateID               bigint,
        TimeID               bigint,
        UserDefinedFields    char(18),
```

```
       CONSTRAINT ACCT_cContactHist_PK
              PRIMARY KEY (AccountID, CellID, PackageID )
);
CREATE INDEX ACCT_cContactHist_IX1 ON ACCT_UA_ContactHistory
(
       CellID
);
CREATE INDEX ACCT_cContactHist_IX2 ON ACCT_UA_ContactHistory
(
       PackageID                    ,
       CellID
);
CREATE TABLE ACCT_UA_DtlContactHist (
       AccountID           varchar(30) NOT NULL,
       TreatmentInstID      bigint NOT NULL,
       ContactStatusID      bigint,
       ContactDateTime      timestamp,
       UpdateDateTime       timestamp,
       UserDefinedFields    char(18),
       DateID               bigint NOT NULL,
       TimeID               bigint NOT NULL
);
CREATE INDEX ACCT_cDtlContHist_IX1 ON ACCT_UA_DtlContactHist
(
       AccountID                    ,
       TreatmentInstID
);
ALTER TABLE ACCT_UA_ResponseHistory
       ADD CONSTRAINT ACCT_cRespHistory_FK2
              FOREIGN KEY (TimeID)
                           REFERENCES UA_Time  (TimeID);
ALTER TABLE ACCT_UA_ResponseHistory
       ADD CONSTRAINT ACCT_cRespHistory_FK4
              FOREIGN KEY (DateID)
                           REFERENCES UA_Calendar  (DateID);
ALTER TABLE ACCT_UA_ResponseHistory
       ADD CONSTRAINT ACCT_cRespHistory_FK3
              FOREIGN KEY (ResponseTypeID)
                           REFERENCES UA_UsrResponseType  (
              ResponseTypeID);
ALTER TABLE ACCT_UA_ResponseHistory
       ADD CONSTRAINT ACCT_cRespHistory_FK1
              FOREIGN KEY (TreatmentInstID)
                           REFERENCES UA_Treatment  (
              TreatmentInstID);
ALTER TABLE ACCT_UA_ContactHistory
       ADD CONSTRAINT ACCT_cContactHist_FK2
              FOREIGN KEY (DateID)
                           REFERENCES UA_Calendar  (DateID);
ALTER TABLE ACCT_UA_ContactHistory
       ADD CONSTRAINT ACCT_cContactHist_FK3
              FOREIGN KEY (TimeID)
                           REFERENCES UA_Time  (TimeID);
ALTER TABLE ACCT_UA_ContactHistory
       ADD CONSTRAINT ACCT_cContactHist_FK1
              FOREIGN KEY (ContactStatusID)
                           REFERENCES UA_ContactStatus  (
              ContactStatusID);
ALTER TABLE ACCT_UA_DtlContactHist
       ADD CONSTRAINT ACCT_cDtlContactH_FK3
              FOREIGN KEY (TimeID)
                           REFERENCES UA_Time  (TimeID);
ALTER TABLE ACCT_UA_DtlContactHist
       ADD CONSTRAINT ACCT_cDtlContactH_FK2
              FOREIGN KEY (DateID)
                           REFERENCES UA_Calendar  (DateID);
ALTER TABLE ACCT_UA_DtlContactHist
```

```
        ADD CONSTRAINT ACCT_cDtlContactH_FK1
               FOREIGN KEY (ContactStatusID)
                             REFERENCES UA_ContactStatus  (
               ContactStatusID);
alter table ACCT_UA_DtlContactHist add RTSelectionMethod int;
alter table ACCT_UA_ResponseHistory add RTSelectionMethod int;
```

The below sample script is used in the runtime time IBM DB2 database to create history staging tables in the Interact for an audience level of type Account.

```
DROP TABLE ACCT_UACI_RHStaging;
DROP TABLE ACCT_UACI_CHOfferAttrib;
DROP TABLE ACCT_UACI_CHStaging;
DROP TABLE ACCT_UACI_UserEventActivities;
DROP TABLE ACCT_UACI_EventPatternState;
CREATE TABLE ACCT_UACI_RHStaging (
       SeqNum               bigint NOT NULL,
       TreatmentCode        varchar(512),
       AccountID           varchar(30),
       ResponseDate         timestamp,
       ResponseType         int,
       ResponseTypeCode     varchar(64),
       Mark                 bigint NOT NULL
                                      DEFAULT 0,
       UserDefinedFields    char(18),
  RTSelectionMethod    int,
       CONSTRAINT iRHStaging_PK1
              PRIMARY KEY (SeqNum)
);
CREATE TABLE ACCT_UACI_CHOfferAttrib (
        ContactID            bigint NOT NULL,
        AttributeID          bigint NOT NULL,
        StringValue          varchar(512),
        NumberValue          float,
        DateTimeValue        timestamp,
        CONSTRAINT ACCT_iCHOfferAttrib_PK
               PRIMARY KEY (ContactID, AttributeID)
 );
 CREATE TABLE ACCT_UACI_CHStaging (
        ContactID            bigint NOT NULL,
        TreatmentCode        varchar(512),
        CampaignID           bigint,
        OfferID              bigint,
        CellID               bigint,
        AccountID           varchar(30),
        ContactDate          timestamp,
        ExpirationDateTime   timestamp,
        EffectiveDateTime    timestamp,
        ContactType          int,
        UserDefinedFields      char(18),
        Mark                 bigint NOT NULL DEFAULT 0,
  RTSelectionMethod    bigint,
        CONSTRAINT ACCT_iCHStaging_PK
               PRIMARY KEY (ContactID)
 );
CREATE TABLE ACCT_UACI_UserEventActivity
(
       SeqNum               bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
       ICID                 bigint NOT NULL,
       ICName               varchar(64) NOT NULL,
       CategoryID           bigint NOT NULL,
       CategoryName         varchar(64) NOT NULL,
       EventID              bigint NOT NULL,
       EventName            varchar(64) NOT NULL,
       TimeID               bigint,
       DateID               bigint,
       Occurrences          bigint NOT NULL,
```

```
       AccountID varchar(30) not null,
       CONSTRAINT iUserEventActivity_PK
              PRIMARY KEY (SeqNum)
);
create table ACCT_UACI_EventPatternState
(
 UpdateTime bigint not null,
 State varchar(1000) for bit data,
 AccountID varchar(30) not null,
        CONSTRAINT iCustomerPatternState_PK
        PRIMARY KEY (AccountID,UpdateTime)
);
 ALTER TABLE ACCT_UACI_CHOfferAttrib
       ADD CONSTRAINT ACCT_iCHOfferAttrib_FK1
              FOREIGN KEY (ContactID)
                            REFERENCES ACCT_UACI_CHStaging  (ContactID);
```

Configuring JMX monitoring for the contact and response history module

Use this procedure to configure JMX monitoring for the contact and response history module. The JMXMP and RMI protocols are supported. Configuring JMX monitoring does not enable security for the contact and response history module. You use the Marketing Platform for the design environment to configure the JMX monitoring.

To use your JMX monitoring tool for the contact and response history module, the default address that is used for:

- The JMXMP protocol is `service:jmx:jmxmp://`*`CampaignServer`*`:`*`port`*`/campaign`.
- The RMI protocol is `service:jmx:rmi:///jndi/rmi://`*`CampaignServer`*`:`*`port`*`/campaign`.

When you view the data in your JMX monitoring tool, the results attributes are organized first by partition and next by audience level.

In Marketing Platform for the design environment, edit the following configuration properties in the `Campaign > monitoring` category.

Configuration property	Setting
`monitorEnabledForInteract`	**True**
port	The port number for the JMX service
protocol	The protocol to use: • **JMXMP** • **RMI** Security is not enabled for the contact and response history module, even if you select the JMXMP protocol.

About cross-session response tracking

Visitors may not always complete a transaction in a single visit to your touchpoint. A customer may add an item to their shopping cart on your web site and not complete the sale until two days later. Keeping the runtime session active indefinitely is not feasible. You can enable cross-session response tracking to track an offer presentation in one session and match it with a response in another session.

Interact cross-session response tracking can match on treatment codes or offer codes by default. You can also configure it to match any custom code of your choice. Cross-session response matches on the available data. For example, your web site includes an offer with a promotional code generated at the time of display for a discount good for one week. A user may add items to their shopping cart, but not complete the purchase until three days later. When you use the `postEvent` call to log an accept event, you can include only the promotional code. Because the runtime cannot find a treatment or offer code to match in the current session, the runtime places the accept event with the available information in a cross-session response (XSessResponse) staging table. The CrossSessionResponse service periodically reads the XSessResponse table and attempts to match the records with the available contact history data. The CrossSessionResponse service matches the promotional code to the contact history and collects all the required data to log a proper response. The CrossSessionResponse service then writes the response to the response staging tables, and if learning is enabled, the learning tables. The contact and response history module then writes the response to the Campaign contact and response history tables. The successful processing of the cross-session response depends on the original contact history records that has been migrated to the Campaign database by the contact history ETL.

Cross-session response tracking data source configuration

Interact cross-session response tracking matches session data from the runtime environment with the Campaign contact and response history. By default, cross-session response tracking matches on treatment code or offer code. You can configure the runtime environment to match on a custom, alternate code.

- If you choose to match on an alternate code, you must define the alternate code in the `UACI_TrackingType` table in the Interact runtime tables.
- The runtime environment must have access to the Campaign contact history tables. This can be by either configuring the runtime environment to have access to the Campaign contact history tables, or by creating a copy of the contact history tables in the runtime environment.

  This access is read-only, and is separate from the contact and response history utility.

  If you create a copy of the tables, it is your responsibility to ensure data in the copy of the contact history is accurate. You can configure the length of time the CrossSessionResponse service retains unmatched responses to match the how often you refresh the data in the copy of the contact history tables using the `purgeOrphanResponseThresholdInMinutes` property. If you are using the contact and response history module, you should coordinate the ETL updates to ensure you have the most current data.

Configuring contact and response history tables for cross-session response tracking

Whether you create a copy of the contact history tables, or use the actual tables in the Campaign system tables, you must perform the following steps to configure the contact and response history tables.

The contact and response history tables must be mapped properly in Campaign prior to performing these steps.

1. Run the `aci_lrnfeature` SQL script in the `interactDT/ddl/acifeatures` directory in the Interact design environment installation directory against the `UA_DtlContactHist` and `UA_ResponseHistory` tables in your Campaign system tables.

This adds the RTSelectionMethod column to the UA_DtlContactHist and UA_ResponseHistory tables. Run the aci_lrnfeature script against these tables for each of your audience levels. Edit the script as necessary to work with the correct table for each of your audience levels.

2. If you want to copy the contact history tables to the runtime environment, do so now.

   If you are creating a copy of the Campaign contact history tables accessible by the runtime environment for cross-session response tracking support, use the following guidelines:

   - Cross-session response tracking requires read-only access to these tables.
   - Cross-session response tracking requires the following tables from the Campaign contact history.
     - UA_DtlContactHist (for each audience level)
     - UA_Treatment

     You must update the data in these tables on a regular basis to ensure accurate response tracking.
3. Run the aci_crhtab SQL script in the ddl directory in the Interact runtime environment installation directory against the contact and response history data source.

   This script creates the UACI_XsessResponse and UACI_CRHTAB_Ver tables.
4. Create a version of the UACI_XsessResponse table for each audience level.

To improve the performance of cross-session response tracking, you may want to limit the amount of contact history data, either by the way in which you copy the contact history data or by configuring a view in to the Campaign contact history tables. For example, if you have a business practice that no offer is valid for longer than 30 days, you should limit the contact history data to the last 30 days. To modify the number of days of contact history data to maintain, open the configuration property **Campaign | partitions | partition*n*| Interact | contactAndResponseHistTracking** and set the value of **daysBackInHistoryToLookupContact**.

You will not see results from cross-session response tracking until the contact and response history module runs. For example, the default processSleepIntervalInMinutes is 60 minutes. Therefore, it may take at least an hour before cross-session responses appear in your Campaign response history.

UACI_TrackingType table

The UACI_TrackingType table is part of the runtime environment tables. This table defines the tracking codes used with cross-session response tracking. The tracking code defines what method the runtime environment uses to match the current offer in a runtime session with the contact and response history.

Column	Type	Description
TrackingCodeType	int	A number representing the tracking code type. This number is referenced by the SQL commands used to match information from the session data to the contact and response history tables.
Name	varchar(64)	The name for the tracking code type. This is passed in to session data using the UACI_TrackingCodeType reserved parameter with the postEvent method.

Column	Type	Description
Description	varchar(512)	A brief description of the tracking code type. This field is optional.

By default, the runtime environment has two tracking code types defined, as shown in the following table. For any alternate code, you must define a unique TrackingCodeType.

TrackingCodeType	Name	Description
1	Treatment Code	UACI Generated Treatment Code
2	Offer Code	UAC Campaign Offer Code

UACI_XSessResponse

The UACI_XSessResponse table is part of the runtime environment tables. This table is used for cross-session response tracking.

One instance of this table for each audience level must exist in the contact and response history data source available for Interact cross-session response tracking.

Column	Type	Description
SeqNumber	bigint	Identifier for the row of data. The CrossSessionResponse service processes all records in the SeqNumber order.
ICID	bigint	Interactive channel ID
AudienceID	bigint	The audience ID for this audience level. The name of this column must match the audience ID defined in Campaign. The sample table contains the column CustomerID.
TrackingCode	varchar(64)	The value that is passed by UACIOfferTrackingCode parameter of the postEvent method.
TrackingCodeType	int	The numeric representation of the tracking code. The value must be a valid entry in the UACI_TrackingType table.
OfferID	bigint	The offer ID as defined in Campaign.
ResponseType	int	The response type for this record. The value must be a valid entry in the UA_UsrResponseType table.
ResponseTypeCode	varchar(64)	The response type code for this record. The value must be a valid entry in the UA_UsrResponseType table.
ResponseDate	datetime	The date of the response.

Column	Type	Description
Mark	bigint	The value of this field identifies the state of the record. • 1 - In process • 2 - Successful • NULL - Retry • -1 - Record has been in the database for more than purgeOrphanResponseThresholdInMinutes minutes. As part of the database administrator's maintenance of this table, you can check this field for records that are not being matched, that is, all records with value of -1. All records with value 2 are automatically removed by the CrossSessionResponse service.
UsrDefinedFields	char(18)	Any custom fields that you want to include when you are matching offer responses to the contact and response history. For example, if you want to match on a promotional code, include a promotional code user-defined field.

Enabling cross-session response tracking

Use this procedure to enable cross-section response tracking.

You must configure the contact and response history module to take full advantage of cross-session response tracking.

To use cross-session response tracking, you must configure the runtime environment to have read-access to the Campaign contact and response history tables. You can read from either the actual Campaign contact and response history tables in the design environment, or a copy of the tables in the runtime environment data sources. Configuring the runtime environment to have read-access to the contact and response history table is separate from any contact and response history module configuration.

If you are matching on something other than treatment code or offer code, you must add it to the UACI_TrackingType table.

1. Create the XSessResponse tables in the contact and response history tables accessible to the runtime environment.
2. Define the properties in the contactAndResponseHistoryDataSource category for the runtime environment.
3. Define the crossSessionResponseTable property for each audience level.
4. Create an OverridePerAudience category for each audience level.

Cross-session response offer matching

By default, cross-session response tracking matches on treatment codes or offer codes. The crossSessionResponse service uses SQL commands to match treatment codes, offer codes, or a custom code from session data to the Campaign contact and response history tables. You can edit these SQL commands to match any customizations you make to your tracking codes, offer codes, or custom codes.

Matching by treatment code

The SQL to match by treatment code must return all the columns in the XSessResponse table for this audience level plus a column called `OfferIDMatch`. The value in the `OfferIDMatch` column must be the `offerId` that goes with the treatment code in the XSessResponse record.

The following is a sample of the default generated SQL command that match treatment codes. Interact generates the SQL to use the correct table names for the audience level. This SQL is used if the `Interact > services > crossSessionResponse > OverridePerAudience > AudienceLevel > TrackingCodes > byTreatmentCode > SQL` property is set to **Use System Generated SQL**.

```
select   distinct treatment.offerId as OFFERIDMATCH,
   tx.*,
            dch.RTSelectionMethod
from     UACI_XSessResponse tx
Left Outer Join UA_Treatment treatment ON tx.trackingCode=treatment.treatmentCode
Left Outer Join UA_DtlContactHist dch ON tx.CustomerID = dch.CustomerID
Left Outer Join UA_ContactHistory ch ON tx.CustomerID = ch.CustomerID
AND treatment.cellID = ch.cellID
AND treatment.packageID=ch.packageID
where   tx.mark=1
and       tx.trackingCodeType=1
```

The values `UACI_XsessResponse`, `UA_DtlContactHist`, `CustomerID`, and `UA_ContactHistory` are defined by your settings in Interact. For example, `UACI_XsessResponse` is defined by the `Interact > profile > Audience Levels > [AudienceLevelName] > crossSessionResponseTable` configuration property.

If you have customized your contact and response history tables, you may need to revise this SQL to work with your tables. You define SQL overrides in the `Interact > services > crossSessionResponse > OverridePerAudience > (AudienceLevel) > TrackingCodes > byTreatmentCode > OverrideSQL` property. If you provide some override SQL, you must also change the `SQL` property to **Override SQL**.

Matching by offer code

The SQL to match by offer code must return all the columns in the XSessResponse table for this audience level plus a column called `TreatmentCodeMatch`. The value in the `TreatmentCodeMatch` column is the Treatment Code that goes with the Offer ID (and Offer Code) in the XSessResponse record.

The following is a sample of the default generated SQL command that match offer codes. Interact generates the SQL to use the correct table names for the audience level. This SQL is used if the `Interact > services > crossSessionResponse > OverridePerAudience > AudienceLevel > TrackingCodes > byOfferCode > SQL` property is set to **Use System Generated SQL**.

```
select   treatment.treatmentCode as TREATMENTCODEMATCH,
   tx.*,
dch.RTSelectionMethod
from     UACI_XSessResponse tx
Left Outer Join UA_DtlContactHist dch ON tx.CustomerID=dch.CustomerID
Left Outer Join UA_Treatment treatment ON tx.offerId = treatment.offerId
Left Outer Join
   (
   select   max(dch.contactDateTime) as maxDate,
      treatment.offerId,
      dch.CustomerID
   from     UA_DtlContactHist dch, UA_Treatment treatment, UACI_XSessResponse tx
```

```
   where  tx.CustomerID=dch.CustomerID
   and tx.offerID = treatment.offerId
   and dch.treatmentInstId = treatment.treatmentInstId
   group by dch.CustomerID,  treatment.offerId
   ) dch_by_max_date ON tx.CustomerID=dch_by_max_date.CustomerID
    and tx.offerId = dch_by_max_date.offerId
where   tx.mark = 1
and       dch.contactDateTime = dch_by_max_date.maxDate
and       dch.treatmentInstId = treatment.treatmentInstId
and       tx.trackingCodeType=2
union
select   treatment.treatmentCode as TREATMENTCODEMATCH,
   tx.*,
   0
from UACI_XSessResponse tx
Left Outer Join UA_ContactHistory ch ON tx.CustomerID =ch.CustomerID
Left Outer Join UA_Treatment treatment ON tx.offerId = treatment.offerId
Left Outer Join
   (
   select   max(ch.contactDateTime) as maxDate,
      treatment.offerId, ch.CustomerID
   from UA_ContactHistory ch, UA_Treatment treatment, UACI_XSessResponse tx
   where  tx.CustomerID =ch.CustomerID
   and tx.offerID = treatment.offerId
   and treatment.cellID = ch.cellID
   and treatment.packageID=ch.packageID
   group by ch.CustomerID,  treatment.offerId
   ) ch_by_max_date ON tx.CustomerID =ch_by_max_date.CustomerID
   and tx.offerId = ch_by_max_date.offerId
   and treatment.cellID = ch.cellID
   and treatment.packageID=ch.packageID
where   tx.mark = 1
   and       ch.contactDateTime = ch_by_max_date.maxDate
   and       treatment.cellID = ch.cellID
   and       treatment.packageID=ch.packageID
   and       tx.offerID = treatment.offerId
   and       tx.trackingCodeType=2
```

The values UACI_XsessResponse, UA_DtlContactHist, CustomerID, and UA_ContactHistory are defined by your settings in Interact. For example, UACI_XsessResponse is defined by the Interact > profile > Audience Levels > [AudienceLevelName] > crossSessionResponseTable configuration property.

If you have customized your contact and response history tables, you may need to revise this SQL to work with your tables. You define SQL overrides in the Interact > services > crossSessionResponse > OverridePerAudience > *(AudienceLevel)* > TrackingCodes > byOfferCode > OverrideSQL property. If you provide some override SQL, you must also change the SQL property to **Override SQL**.

Matching by alternate code

You can define an SQL command to match by some alternate code of your choice. For example, you could have promotional codes or product codes separate from offer or treatment codes.

You must define this alternate code in the UACI_TrackingType table in the Interact runtime environment tables.

You must provide SQL or a stored procedure in the Interact > services > crossSessionResponse > OverridePerAudience > *(AudienceLevel)* > TrackingCodes > byAlternateCode > OverrideSQL property which returns all the columns in the XSessResponse table for this audience level plus the columns

TreatmentCodeMatch and OfferIDMatch. You may optionally return the offerCode in place of OfferIDMatch (in the form of offerCode1, offerCode2, ... offerCode*N* for N part offer codes). The values in the TreatmentCodeMatch column and OfferIDMatch column (or offer code columns) must correspond to the TrackingCode in the XSessResponse record.

For example, the following SQL pseudo code matches on the AlternateCode column in the XSessResponse table.

```
Select m.TreatmentCode as TreatmentCodeMatch, m.OfferID as OfferIDMatch, tx.*
From MyLookup m, UACI_XSessResponse tx
Where m.customerId = tx.customerId
And m.alternateCode = tx.trackingCode
And tx.mark=1
And tx.trackingCodeType = <x>
```

Where <*x*> is the tracking code defined in the UACI_TrackingType table.

Using a database load utility with the runtime environment

By default, the runtime environment writes contact and response history data from session data into staging tables. On a very active production system, however, the amount of memory required to cache all the data before runtime can write it to the staging tables may be prohibitive. You can configure runtime to use a database load utility to improve performance.

When you enable a database load utility, instead of holding all contact and response history in memory before writing to the staging tables, runtime writes the data to a staging file. You define the location of the directory containing the staging files with the externalLoaderStagingDirectory property. This directory contains several subdirectories. The first subdirectory is the runtime instance directory, which contains the contactHist and respHist directories. The contactHist and respHist directories contain uniquely named subdirectories in the format of *audienceLevelName.uniqueID.currentState*, which contain the staging files.

Current State	Description
CACHE	Contents of directory currently being written to a file.
READY	Contents of directory ready to be processed.
RUN	Contents of directory currently being written to the database.
PROCESSED	Contents of directory have been written to the database.
ERROR	An error occurred while writing the contents of directory to the database.
ATTN	Contents of directory need attention. That is, you may need to take some manual steps to complete writing the contents of this directory to the database.
RERUN	Contents of directory ready to be written to the database. You should rename a directory from ATTN or ERROR to RERUN after you have corrected the problem.

You can define the runtime instance directory by defining the interact.runtime.instance.name JVM property in the application server startup script. For example, you could add -Dinteract.runtime.instance.name=instance2 to your web application server startup script. If not set, the default name is DefaultInteractRuntimeInstance.

The samples directory contains sample files to assist you with writing your own database load utility control files.

Enabling a database load utility with runtime environment

Use this procedure to enable a database load utility with the runtime environment.

You must define any command or control files for your database load utility before you configure runtime environment to use them. These files must exist in the same location on all runtime servers in the same server group.

Interact provides sample command and control files in the loaderService directory in your Interact runtime server installation.

1. Confirm that the runtime environment user has login credentials for the runtime tables data source that is defined in Interact > general > systemTablesDataSource in your configuration properties.
2. Define the Interact > general > systemTablesDataSource > loaderProperties configuration properties.
3. Define the Interact > services >externalLoaderStagingDirectory property.
4. Revise the Interact > services > responseHist > fileCache configuration properties, if necessary.
5. Revise the Interact > services > contactHist > fileCache configuration properties, if necessary.
6. Restart the runtime server.

Event pattern ETL process

To process large amounts of IBM Interact event pattern data and to make that data available for queries and reporting purposes, you can install a stand-alone Extract, Transform, Load (ETL) process on any supported server for optimal performance.

In Interact, all event pattern data for a given AudienceID is stored as a single collection in the runtime database tables. The AudienceID and pattern state information is stored as a Binary Large Object (BLOB). To perform any SQL queries or reporting based on event patterns, this new ETL process is necessary to break up the object into tables into a target database. To accomplish this, the stand-alone ETL process takes event pattern data from the Interact runtime database tables, processes it on the schedule you specify, and stores it in the target database where it is available for SQL queries or additional reporting.

In addition to moving and transforming event pattern data to the target database, the stand-alone ETL process also synchronizes the data in the target database with the most current information in your Interact runtime database. For example, if you delete an event pattern in the Interact runtime, that event pattern's processed data is removed from the target database the next time the ETL process runs. Event pattern state information is kept up to date as well. So the information stored about event patterns in the target database is solely current data, not historical information.

Running the stand-alone ETL process

When you launch the stand-alone ETL process on a server, it runs continuously in the background until stopped. The process follows the instructions in the Marketing Platform configuration properties to determine frequency, database connections, and other details during its operation.

Before you run the stand-alone ETL process, ensure that you complete the following tasks:

- You must have the permissions of an Interact Admin user role.
- You must have installed the process on a server, and configured both the files on the server and in the Marketing Platform correctly for your configuration.

Note:

If you are running the ETL process on Microsoft Windows for a language other than US English, use `chcp` at the command prompt to set the code page for the language you are using. For example, you might use any of the following codes: ja_jp=932, zh_cn=936, ko_kr=949, ru_ru=1251 and for de_de, fr_fr, it_it, es_es, pt_br, use 1252. To ensure proper character display, use the `chcp` command in the Windows command prompt prior to launching the ETL process.

After you have installed and configured the stand-alone ETL process, you are ready to launch the process.

1. Open a command prompt on the server where the ETL process is installed.
2. Navigate to the `<Interact_home>/PatternStateETL/bin` directory that contains the executable files for the ETL process.
3. Run the `command.bat` file (on Microsoft Windows) or `command.sh` file (on UNIX-like operating systems) with the following parameters:
   - `-u <username>`. This value must be a valid Marketing Platform user, and you must have configured that user with access to the **TargetDS** and **RuntimeDS** datasources that the ETL process will use.
   - `-p <password>`. Replace *<password>* with the password matching the user you specified. If the password for this user is blank, specify two double quotes (as in `-p ""`). The password is optional when you run the command file; if you omit the password with the command, you are prompted to enter it when the command runs.
   - `-c <profileName>`. Replace *<profileName>* with the exact name you specified in the Marketing Platform in the **Interact | PatternStateETL** configuration you created.

     The name you enter here must match the value you specified in the **New category name** field when you created the configuration.
   - `start`. The start command is required to start the process.

   The complete command to start the process would therefore take the following form:

   ```
   command.bat -u <username> -p <password> -c <profileName> start
   ```

The stand-alone ETL process runs, and continues to run in the background until you stop the process or until the server is restarted.

Note:

The first time that you run the process, the accumulated event pattern data may take a considerable amount of time to run. Subsequent times that the process runs will work with only the most recent set of event pattern data and takes less time to complete.

Be aware that you can also provide the `help` argument to the `command.bat` or `command.sh` file to see all available options, as in the following example:

```
command.bat help
```

Stopping the stand-alone ETL process

When you launch the stand-alone ETL process on a server, it runs continuously in the background until stopped.

1. Open a command prompt on the server where the ETL process is installed.
2. Navigate to the `<Interact_home>/PatternStateETL/bin` directory that contains the executable files for the ETL process.
3. Run the `command.bat` file (on Microsoft Windows) or `command.sh` file (on UNIX-like operating systems) with the following parameters:
   - `-u <username>`. This value must be a valid Marketing Platform user, and you must have configured that user with access to the **TargetDS** and **RuntimeDS** data sources that the ETL process will use.
   - `-p <password>`. Replace *<password>* with the password matching the user you specified. If the password for this user is blank, specify two double quotes (as in `-p ""`). The password is optional when you run the command file; if you omit the password with the command, you are prompted to enter it when the command runs.
   - `-c <profileName>`. Replace *<profileName>* with the exact name you specified in the Marketing Platform in the **Interact | PatternStateETL** configuration you created.

     The name you enter here must match the value you specified in the **New category name** field when you created the configuration.
   - `stop`. The stop command is required to stop the process. If you use this command, any ongoing ETL operation will complete before the process shuts down.

     To shut down the ETL process without waiting for any ongoing operations to complete, use `forcestop` instead of `stop`.

   The complete command to start the process would therefore take the following form:

   ```
   command.bat -u <username> -p <password> -c <profileName> stop
   ```

The stand-alone ETL process stops.

Chapter 4. Offer serving

You can configure Interact in many ways to enhance how it selects offers to present. The following sections describe these optional features in detail.

Offer eligibility

The purpose of Interact is to present eligible offers. Simply, Interact presents the most optimal among the eligible offers, based on the visitor, the channel, and the situation.

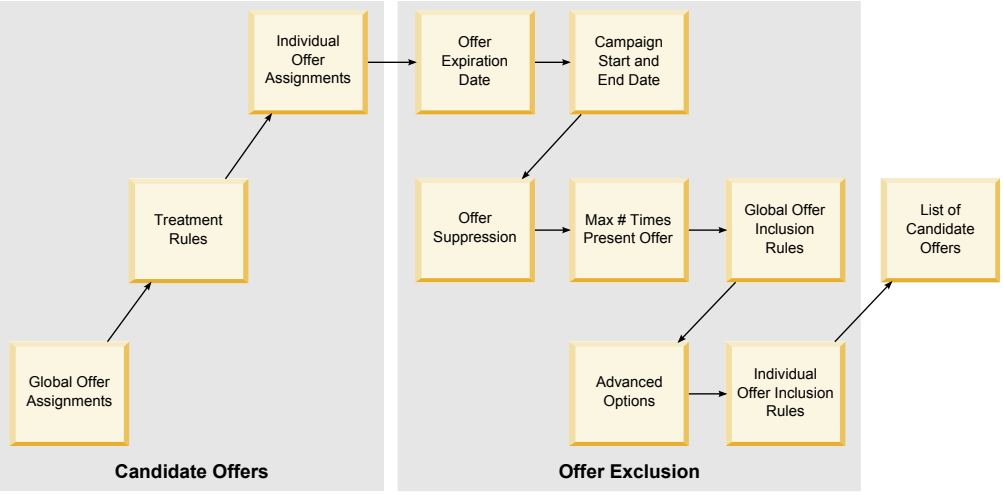
Treatment rules are only the start of how Interact determines which offers are eligible for a customer. Interact has several optional features which you can implement to enhance how the runtime environment determines which offers to present. None of these features guarantee that an offer is presented to a customer. These features influence the probability that an offer is eligible to be presented to a customer. You can use as many or as few of these features as you need to implement the best solution for your environment.

There are three main areas where you can influence offer eligibility: generating the list of candidate offers, determining the marketing score, and learning.

Generating a list of candidate offers

Generating a list of candidate offers has two major stages. The first stage is generating a list of all possible offers for which the customer may be eligible. The second stage is filtering out any offer for which the customer is no longer eligible. There are several places in both stages where you can influence the generation of the candidate offer list.

This diagram shows the stages of the candidate offer list generation. The arrows show the order of precedence. For example, if an offer passes the **Max # of times to present an offer** filter, but fails the **Global offer inclusion rules** filter, the runtime environment excludes the offer.



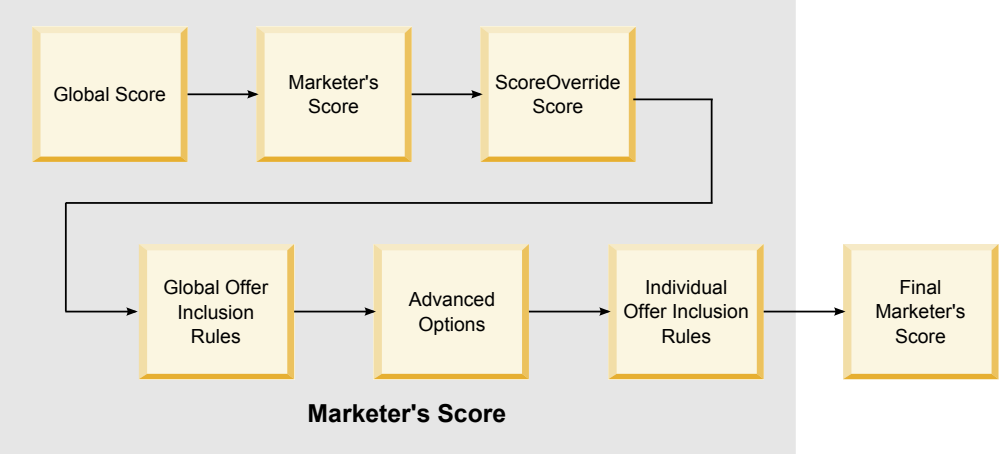
- **Global offer assignments** - You can define global offers by audience level using the global offers table.

- **Treatment rules** - The basic method to define offers by segment by interaction point using the interaction strategy tab.
- **Individual offer assignments** - You can define specific offer assignments by customer using the score override table.
- **Offer expiration date** - When you create an offer in Campaign, you can define an expiration date. If the expiration date for an offer has passed, the runtime environment excludes the offer.
- **Campaign start and end date** - When you create a campaign in Campaign, you can define a start and end date for the campaign. If the start date for the campaign has not occurred or the end date for the campaign has passed, the runtime environment excludes the offer.
- **Offer suppression** - You can define offer suppression for specific audience members using the offer suppression table.
- **Max # times to present an offer** - When you define an interactive channel, you define the maximum number of times to present an offer to a customer per session. If the offer has already been presented this number of times, the runtime environment excludes the offer.
- **Global offer inclusion rules** - You can define a boolean expression to filter offers on an audience level using the global offers table. If the result is false, the runtime environment excludes the offer.
- **Advanced options** - You can use the **Consider this rule eligible if the following expression is true** advanced option in a treatment rule to filter offers on a segment level. If the result is false, the runtime environment excludes the offer.
- **Individual offer inclusion rules** - You can define a boolean expression to filter offers on a customer level using the score override table. If the result is false, the runtime environment excludes the offer.

Calculate the marketing score

There are many ways to influence (by using a calculation) or override the marketing score.

This diagram shows the different stages where you can influence or override the marketing score.



The arrows show the order of precedence. For example, if you define an expression to determine the marketing score in the Advanced Options for a treatment rule and define an expression in the score override table, the expression in the score override table takes precedence.

- **Global score** - You can define a score per audience level using the global offers table.
- **Marketer's score** - You can define a score per segment using the slider in a treatment rule.
- **Score Override score** - You can define a score per customer using the score override table.
- **Global offer inclusion rules** - You can define an expression which calculates a score per audience level using the global offers table.
- **Advanced Options** - You can define an expression which calculates a score per segment using the **Use the following expression as the marketing score** advanced option in a treatment rule.
- **Score override offer inclusion rules** - You can define an expression which calculates a score per customer using the score override table.

Influencing learning

If you are using the Interact built-in learning module, you can influence the learning output beyond the standard learning configurations such as the list of learning attributes or the confidence level. You can override components of the learning algorithm while using the remaining components.

You can override learning using the `LikelihoodScore` and `AdjExploreScore` columns of the default offers and score override tables. You can add these columns to the default offers and score override tables using the `aci_scoringfeature` feature script. To properly use these overrides, you need a thorough understanding of Interact built-in learning.

The learning module takes the list of candidate offers and the marketing score per candidate offer and uses them in the final calculations. The offer list is used with the learning attributes to calculate the likelihood (accept probability) that the customer will accept the offer. Using these probabilities and the historical number of presentations to balance between exploration and exploitation, the learning algorithm determines the offer weight. Finally, the built-in learning takes the offer weight, multiplies it by the final marketing score and returns a final score. The offers are sorted by this final score.

Suppress offers

You can configure the runtime environment to suppress offers.

There are several ways in which the runtime environment suppresses an offer:

- The **Maximum # of times to show any offer during a single visit** element of an interactive channel.

  You define the **Maximum # of times to show any offer during a single visit** when you create or edit an interactive channel.
- The use of an offer suppression table.

  You create an offer suppression table in your profile database.
- Offers whose expiration date has passed.
- Offers from expired campaigns.
- Offers excluded because they do not pass an offer inclusion rule (treatment rule advanced option).
- Offers already explicitly accepted or rejected in a Interact session. If a customer explicitly accepts or rejects an offer, that offer is suppressed during the session.

Enabling offer suppression

Use this procedure to enable offer suppression.

You can configure Interact to reference a list of suppressed offers.

1. Create an `offerSuppressionTable`, a new table for every audience that contains the audience ID and the offer ID.
2. Set the `enableOfferSuppressionLookup` property to **true**.
3. Set the `Interact > profile > offerSuppressionTable` property to the name of the offer suppression table for the appropriate audience.

Offer suppression table

The offer suppression table enables you to suppress an offer for a specific audience ID. For example, if your audience is Customer, you can suppress an offer for the customer John Smith. A version of this table for at least one audience level must exist in your production profile database. You can create a sample offer suppression table, `UACI_Blacklist` by running the `aci_usrtab` SQL script against your profile database. The `aci_usrtab` SQL script is located in the `ddl` directory in your runtime environment installation directory.

You must define the `AudienceID` and `OfferCode1` fields for each row. You can add additional columns if your Audience ID or Offer Code consists of multiple columns. These columns must match the column names defined in Campaign. For example if you define the audience `Customer` by the fields `HHold_ID` and `MemberNum`, you must add `HHold_ID` and `MemberNum` to the offer suppression table.

Name	Description
AudienceID	(Required) The name of this column must match the name of the column defining the audience ID in Campaign. If your audience ID consists of multiple columns, you can add them to this table. Each row must contain the audience ID to which you assign the default offer, for example, `customer1`.
OfferCode1	(Required) The offer code for the offer you are overriding. If your offer codes are made of multiple fields, you can add the additional columns, for example `OfferCode2`, and so on.

Global offers and individual assignments

You can configure the runtime environment to assign specific offers beyond the treatment rules configured on the Interaction Strategy tab. You can define global offers for any member of an audience level and individual assignments for specific audience members. For example, you can define a global offer for all households to see when no others are available, and then create an individual offer assignment for the specific Smith household.

You can constrain both global offers and individual assignments by zone, cell, and offer inclusion rules. Both global offers and individual assignments are configured by adding data to specific tables in your production profile database.

For global offers and individual assignments to function properly, all referenced cell and offer codes must exist in the deployment. To ensure the required data is available, you must configure default cell codes and the `UACI_ICBatchOffers` table.

Defining the default cell codes

If you use the default offers or score override tables for global or individual offer assignments, you must define default cell codes. The `DefaultCellCode` is used when there is no defined cell code in a particular row in the default offers or score override tables. Reporting uses this default cell code.

The `DefaultCellCode` must match the cell code format that is defined in Campaign. This cell code is used for all offer assignments that appear in reporting.

If you define unique default cell codes, you can easily identify offers that are assigned by the default offers or score override tables.

Define the `DefaultCellCode` property for each audience level and table type in the `IndividualTreatment` category.

Defining offers not used in a treatment rule

If you use the default offers or score override tables, you must ensure that all offer codes exist in the deployment. If you know that all offers you use in the default offers or score override tables are used in your treatment rules, the offers exist in the deployment. However, any offer that is not used in a treatment rule must be defined in the `UACI_ICBatchOffers` table.

The `UACI_ICBatchOffers` table exists in the Campaign system tables.

Populate the `UACI_ICBatchOffers` table with offer codes that you use in the default offer or score override tables. The table has the following format:

Column Name	Type	Description
ICName	varchar(64)	The name of the interactive channel the offer is associated with. If you are using the same offer with two different interactive channels, you must provide a row for each interactive channel.
OfferCode1	varchar(64)	The first part of the offer code.
OfferCode2	varchar(64)	The second part of the offer code.
OfferCode3	varchar(64)	The third part of the offer code.
OfferCode4	varchar(64)	The fourth part of the offer code.
OfferCode5	varchar(64)	The fifth part of the offer code

About the global offers table

The global offers table enables you to define treatments at the audience level. For example, you can define a global offer for every member of the audience Household.

You can define global settings for the following elements of Interact offer serving.

- Global offer assignment
- Global marketer's score, by a number or by an expression
- Boolean expression to filter offers
- Learning probability and weight, if you are using Interact Built-in Learning
- Global learning override

Assigning global offers

Use this procedure to configure the runtime environment to assign global offers for an audience level, beyond anything that is defined in treatment rules.

1. Create a table that is called `UACI_DefaultOffers` in your profile database.

   To create the `UACI_DefaultOffers` table with the correct columns, use the `aci_usrtab` ddl file.
2. Set the `Interact > profile > enableDefaultOfferLookup` property to **true**.

Global offer table

The global offer table must exist in your profile database. You can create the global offer table, `UACI_DefaultOffers` by running the `aci_usrtab` SQL script against your profile database.

The `aci_usrtab` SQL script is located in the `ddl` directory in your runtime environment installation directory .

You must define the `AudienceLevel`, and `OfferCode1` fields for each row. The other fields are optional to constrain your offer assignments further or influence the built-in learning at the audience level.

For best performance, you should create an index on this table on the audience level column.

Name	Type	Description
AudienceLevel	varchar(64)	(Required) The name of the audience level you assign the default offer to, for example, `customer` or `household`. This name must match the audience level as defined in Campaign.
OfferCode1	varchar(64)	(Required) The offer code for the default offer. If your offer codes are made of multiple fields, you can add the additional columns, for example `OfferCode2` and so on. If you are adding this offer to provide a global offer assignment, you must add this offer to the `UACI_ICBatchOffers` table.
Score	float	A number to define the marketing score for this offer assignment.
OverrideTypeID	int	If set to `1`, if the offer does not exist in the candidate list of offers, add this offer to the list as well as using any score data for the offer. In general, use 1 to provide global offer assignments. If set to `0`, *null*, or any number other than 1, use any data for the offer only if the offer exists in the candidate list of offers. In most cases, a treatment rule or individual assignment will override this setting.

Name	Type	Description
Predicate	varchar(4000)	You can enter expressions in this column as for advanced options for treatment rules. You can use the same variables and macros available to you when writing advanced options for treatment rules. The behavior of this column depends on the value in the `EnableStateID` column. • If the `EnableStateID` is `2`, this column works the same as **Consider this rule eligible if the following expression is true** option in the advanced options for treatment rules to constrain this offer assignment. This column must contain a boolean expression, and resolve to true to include this offer. If you accidentally define an expression that resolves to a number, any non-zero number is considered true and zero is considered false. • If the `EnableStateID` is `3`, this column works the same as **Use the following expression as the marketing score** option in the advanced options for treatment rules to constrain this offer. This column must contain an expression that resolves to a number. • If the `EnableStateID` is `1`, Interact ignores any value in this column.
FinalScore	float	A number to override the final score used to order the final list of returned offers. This column is used if you have enabled the built-in learning module. You can implement your own learning to use this column.
CellCode	varchar(64)	The cell code for a deployed interactive segment to which you want to assign this default offer. If your cell codes are made of multiple fields, you can add the additional columns. You must provide a cell code if `OverrideTypeID` is `0` or null. If you do not include a cell code, the run time environment ignores this row of data. If the `OverrideTypeID` is `1`, you do not have to provide a cell code in this column. If you do not provide a cell code, the runtime environment uses the cell code defined in the `DefaultCellCode` property for this audience level and table for reporting purposes.
Zone	varchar(64)	The name of the zone to which you want this offer assignment to apply. If NULL, this applies to all zones.

Name	Type	Description
EnableStateID	int	The value in this column defines the behavior of the `Predicate` column. • **1** - Do not use the `Predicate` column. • **2** - Use `Predicate` as a boolean to filter the offer. This follows the same rules as the **Consider this rule eligible if the following expression is true** advanced option in a treatment rule. • **3** - Use `Predicate` to define the marketer's score. This follows the same rules as the **Use the following expression as the marketing score** advanced option in a treatment rule. Any row where this column is Null or any value other than 2 or 3 ignores the `Predicate` column.
LikelihoodScore	float	This column is used only to influence built-in learning. You can add this column with the `aci_scoringfeature` ddl.
AdjExploreScore	float	This column is used only to influence built-in learning. You can add this column with the `aci_scoringfeature` ddl.

About the score override table

The score override table allows you to define treatments on an audience ID or individual level. For example, if your audience level is Visitor, you can create overrides for specific visitors.

You can define overrides for the following elements of Interact offer serving.

- Individual offer assignment
- Individual marketer's score, by a number or by an expression
- Boolean expression to filter offers
- Learning probability and weight, if you are using Built-in Learning
- Individual learning override

Configuring score overrides

You can configure Interact to use a score that is generated from a modeling application instead of the marketing score.

1. Create a score override table for each audience level for which you want to provide overrides.

   To create a sample score override table with the correct columns, use the `aci_usrtab` ddl file.
2. Set the `Interact > Profile > enableScoreOverrideLookup` property to **true**.
3. Set the `scoreOverrideTable` property to the name of the score override table for each audience level for which you want to provide overrides.

   You do not need to provide a score override table for every audience level.

Score override table

The score override table must exist in your production profile database. You can create a sample score override table, `UACI_ScoreOverride` by running the `aci_usrtab` SQL script against your profile database.

The aci_usrtab SQL script is located in the ddl directory in your runtime environment installation directory.

You must define the *AudienceID*, OfferCode1, and Score fields for each row. The values in the other fields are optional to constrain your individual offer assignments further or provide score override information for the built-in learning.

Name	Type	Description
AudienceID	varchar(64)	(Required) The name of this column must match the name of the column defining the audience ID in Campaign. The sample table created by the aci_usrtab ddl file create this column as the CustomerID column. If your audience ID consists of multiple columns, you can add them to this table. Each row must contain the audience ID to which you assign the individual offer, for example, customer1. For best performance, you should create an index on this column.
OfferCode1	varchar(64)	(Required) The offer code for the offer. If your offer codes are made of multiple fields, you can add the additional columns, for example OfferCode2 and so on. If you are adding this offer to provide an individual offer assignment, you must add this offer to the UACI_ICBatchOffers table.
Score	float	A number to define the marketing score for this offer assignment.
OverrideTypeID	int	If set to 0 or *null* (or any number other than 1), use any data for the offer only if the offer exists in the candidate list of offers. In general, use 0 to provide score overrides. You must provide a cell code If set to 1, if the offer does not exist in the candidate list of offers, add this offer to the list as well as using any score data for the offer. In general, use 1 to provide individual offer assignments.

Name	Type	Description
Predicate	varchar(4000)	You can enter expressions in this column as for advanced options for treatment rules. You can use the same variables and macros available to you when writing advanced options for treatment rules. The behavior of this column depends on the value in the `EnableStateID` column. • If the `EnableStateID` is `2`, this column works the same as **Consider this rule eligible if the following expression is true** option in the advanced options for treatment rules to constrain this offer assignment. This column must contain a boolean expression, and resolve to true to include this offer. If you accidentally define an expression that resolves to a number, any non-zero number is considered true and zero is considered false. • If the `EnableStateID` is `3`, this column works the same as **Use the following expression as the marketing score** option in the advanced options for treatment rules to constrain this offer. This column must contain an expression that resolves to a number. • If the `EnableStateID` is `1`, Interact ignores any value in this column.
FinalScore	float	A number to override the final score used to order the final list of returned offers. This column is used if you have enabled the built-in learning module. You can implement your own learning to use this column.
CellCode	varchar(64)	The cell code for an interactive segment to which you want to assign this offer. If your cell codes are made of multiple fields, you can add the additional columns. You must provide a cell code if `OverrideTypeID` is 0 or null. If you do not include a cell code, the run time environment ignores this row of data. If the `OverrideTypeID` is 1, you do not have to provide a cell code in this column. If you do not provide a cell code, the runtime environment uses the cell code defined in the `DefaultCellCode` property for this audience level and table for reporting purposes.
Zone	varchar(64)	The name of the zone to which you want this offer assignment to apply. If NULL, this applies to all zones.

Name	Type	Description
EnableStateID	int	The value in this column defines the behavior of the `Predicate` column. • **1** - Do not use the `Predicate` column. • **2** - Use `Predicate` as a boolean to filter the offer. This follows the same rules as the **Consider this rule eligible if the following expression is true** advanced option in a treatment rule. • **3** - Use `Predicate` to define the marketer's score. This follows the same rules as the **Use the following expression as the marketing score** advanced option in a treatment rule. Any row where this column is Null or any value other than 2 or 3 ignores the `Predicate` column.
LikelihoodScore	float	This column is used only to influence built-in learning. You can add this column with the `aci_scoringfeature` ddl.
AdjExploreScore	float	This column is used only to influence built-in learning. You can add this column with the `aci_scoringfeature` ddl.

Interact built-in learning overview

While you do everything you can to ensure that you propose the right offers to the right segments, you can always learn something from actual selections of your visitors. The actual behavior of your visitors should influence your strategy. You can take response history and run it through some modeling tools to get a score which you can include in your interactive flowcharts.

However, this data is not real-time.

Interact provides two options for you to learn from your visitor's actions in real time:

- Built-in learning module - The runtime environment has a Naive Bayesian-based learning module. This module monitors customer attributes of your choosing and uses that data to help select which offers to present.
- Learning API - The runtime environment also has a learning API for you to write your own learning module.

You do not have to use learning. By default, learning is disabled.

Interact learning module

The Interact learning module monitors visitor's responses to offers and visitor attributes.

Learning module modes

The learning module has two general modes:

- Exploration - the learning module serves offers in order so it can gather enough response data to optimize the estimation that is used during the exploitation mode. Offers served during exploration do not necessarily reflect the optimal choice.

- Exploitation - after enough data is collected by the exploration phase, the learning module uses the probabilities to help select the offers to present.

The learning module uses two properties to alternates between exploration mode and exploitation mode. The two properties are:

- a confidence level that you configure with the `confidenceLevel` property.
- a probability that the learning module presents a random offer that you configure with the `percentRandomSelection` property.

Confidence level property

You set the `confidenceLevel` to a percentage that represents how sure (or confident) the learning module must be before its scores for an offer are used in arbitration. At first, when the learning module has no data to work from, the learning module relies entirely upon the marketing score. After every offer is presented as many times as defined by the `minPresentCountThreshold`, the learning module enters the exploration mode. Without much data to work with, the learning module is not confident that the percentages it calculates are correct. Therefore, it stays in the exploration mode.

The learning module assigns weights to each offer. To calculate the weights, the learning module uses a formula that takes in as input the configured confidence level, the historical acceptance data, and the current session data. The formula inherently balances between exploration and exploitation, and returns the appropriate weight.

Random selection property

To ensure that the system is not biased toward the offers that perform best during early stages, Interact presents a random offer the `percentRandomSelection` percent of the time. This random offer percentage forces the learning module to recommend offers other than the most successful to determine whether other offers would be more successful if they had greater exposure. For example, if you configure `percentRandomSelection` to 5, then 5% of the time the learning module presents a random offer and adds the response data to its calculations.

You can set the **% Random** to specify the change that the returned offer is randomly selected, without considering scores, for each zone on the Interaction Points tab of the Interactive Channel window.

How the learning module determines offers

The learning module determines which offers are presented in the following way.

1. Calculates the probability that a visitor selects an offer.
2. Calculates the offer weight by using the probability from step 1 and determines whether to be in exploration or exploitation mode.
3. Calculates a final score for each offer by using the marketing score and the offer weight from step 2.
4. Sorts the offers by the scores that are determined in step 3 and returns the requested number of top offers.

For example, the learning module determines that a visitor is 30% likely to accept offer A and 70% likely to accept offer B and to exploit this information. From the treatment rules, the marketing score for offer A is 75 and 55 for offer B. However,

the calculations in step 3 makes the final score for offer B higher than offer A, therefore, the runtime environment recommends offer B.

Weight factor properties

Learning is also based on the `recencyWeightingFactor` property and the `recencyWeightingPeriod` property. These properties let you to add more weight to more recent data than older data. The `recencyWeightingFactor` is the percentage of weight to give to the recent data. The `recencyWeightingPeriod` is the length of time that is recent. For example, you configure the `recencyWeightingFactor` to 0.30 and the `recencyWeightingPeriod` to 24. These settings mean that the previous 24 hours of data are 30% of all data considered. For a week's worth of data, all of the data averaged across the first six days is 70% of the data, and the last day is 30% of the data.

Staging table data written

Every session writes the following data to a learning staging table:

- Offer contact
- Offer acceptance
- Learning attributes

At a configurable interval, an aggregator reads the data from the staging table, compiles it, and writes it to a table. The learning module reads this aggregated data and uses it in calculations.

Enabling the learning module

All runtime servers have a built-in learning module. By default, this learning module is disabled. You the enable learning module by changing a configuration property.

In Marketing Platform for the runtime environment, edit the following configuration properties in the `Interact > offerserving` category.

Configuration property	Setting
`optimizationType`	**BuiltInLearning**

Learning attributes

The learning module learns using visitor attributes and offer acceptance data. You can select which visitor attributes you monitor. These visitor attributes can be anything within a customer profile, including some event parameter you collect in real time.

Attributes from dimensional tables are not supported in learning.

While you can configure any number of attributes to monitor, IBM recommends that you configure no more than ten learning attributes between the static and dynamic learning attributes, as well as follow these guidelines.

- Select independent attributes.

  Do not select attributes that are similar. For example, if you create an attribute called HighValue, and that attribute is defined by a calculation based on salary, do not select both HighValue and Salary. Similar attributes do not help the learning algorithm.

- Select attributes with discrete values.

  If an attribute has value ranges, you must select an exact value. For example, if you want to use salary as an attribute, you should give each salary range a specific value, the range 20,000-30,000 should be A, 30,001-40,000 should be B, and so on.You can also define bins in interact and learning system will automatically do the mapping
- Limit the number of attributes you track so you do not impede performance.

  The number of attributes you can track depends on your performance requirements and your Interact installation. If you can, use another modeling tool (such as PredictiveInsight) to determine the top ten predictive attributes. You can configure the learning module to automatically prune attributes that are not predictive, but that also has a performance cost.

You can manage performance by defining both the number of attributes you monitor and the number of values per attribute you monitor. The `Campaign > partitions > partition1 > Interact > learning > maxAttributeNames` property defines the maximum number of visitor attributes you track. The `maxAttributeValues` property defines the maximum number of values you track per attribute. All other values are assigned to a category defined by the value of the `otherAttributeValue` property. However, the learning engine only tracks the first values it encounters. For example, you are tracking the visitor attribute eye color. You are only interested in the values blue, brown, and green, so you set `maxAttributeValues` to 3. However, the first three visitors have the values blue, brown, and hazel. This means that all visitors with green eyes are assigned the `otherAttributeValue`.

You can also use dynamic learning attributes which enable you to define your learning criteria more specifically. Dynamic learning attributes let you learn on the combination of two attributes as a single entry. For example consider the following profile information.

Visitor ID	Card Type	Card Balance
1	Gold Card	$1,000
2	Gold Card	$9,000
3	Bronze Card	$1,000
4	Bronze Card	$9,000

If you use standard learning attributes, you can only learn on card type and balance individually. Visitors 1 and 2 will be grouped together same based on Card Type, and visitors 2 and 4 grouped based on Card Balance. This may not be an accurate predictor of offer acceptance behavior. If Gold Card holders tend to have higher balances, the behavior of Visitor 2 may be radically different than Visitor 4, which would skew the standard learning attributes. However, if you use dynamic learning attributes, each of these visitors is learned on individually and the predictions will be more accurate.

If you use dynamic learning attributes, and the visitor has two valid values for an attribute, the learning module selects the first value it finds.

If you set the `enablePruning` property to `yes`, the learning module algorithmically determines which attributes are not predictive and ceases to consider those attributes when calculating weights. For example, if you are tracking an attribute representing hair color, and the learning module determines that there is no

pattern to accepting an offer based on the visitor's hair color, the learning module ceases to consider the hair color attribute. Attributes are re-evaluated every time the learning aggregation process runs (defined by the `aggregateStatsIntervalInMinutes` property). Dynamic learning attributes are also pruned.

Defining a learning attribute

Use this procedure to define a learning attribute.

You can configure up to the `maxAttributeNames` number of visitor attributes.

The *`(learningAttributes)`* is a template to create new learning attributes. You must enter a new name for each attribute. You cannot create two categories with the same name

In Marketing Platform for the design environment, edit the following configuration properties in the `Campaign > partitions > partition`*n* `> Interact > learning` category.

Configuration property	Setting
`attributeName`	The `attributeName` must match the name of a name-value pair in the profile data. This name is case-insensitive.

Define dynamic learning attributes

To define dynamic learning attributes, you must populate the `UACI_AttributeList` table in the Learning data source.

All columns in this table have the type of varchar(64).

Column	Description
`AttributeName`	The name of the dynamic attribute upon which you want to learn. This value must be an actual value possible in the `AttributeNameCol`.
`AttributeNameCol`	The fully qualified column name (hierarchical structure, starting from profile table) where the `AttributeName` can be found. This column name does not have to be a standard learning attribute.
`AttributeValueCol`	The fully qualified column name (hierarchical structure, starting from profile table) where the associated value for the `AttributeName` can be found.

For example, consider the following profile table and its associated dimension table.

Table 6. MyProfileTable

VisitorID	KeyField
1	Key1
2	Key2
3	Key3
4	Key4

Table 7. MyDimensionTable

KeyField	CardType	CardBalance
Key1	Gold Card	1000
Key2	Gold Card	9000
Key3	Bronze Card	1000
Key4	Bronze Card	9000

The following is a sample `UACI_AttributeList` table matching on card type and balance.

Table 8. UACI_AttributeList

AttributeName	AttributeNameCol	AttributeValueCol
Gold Card	MyProfileTable.MyDimensionTable.CardType	MyProfileTable.MyDimensionTable.CardBalance
Bronze Card	MyProfileTable.MyDimensionTable.CardType	MyProfileTable.MyDimensionTable.CardBalance

Interact AutoBinning

In Interact, the built-in learning algorithm works partly by saving and analysing the values of profile attributes at the time offers were contacted and responded. Some attributes may have virtually unlimited number of unique values. However, due to limited resources in an Interact system, you can save only a small number of them. In addition, often it is more reasonable to do the analysis based on the ranges of the values. You can use this feature is to create such bins in Interact and the learning sub-system will automatically do the mapping.

You can create the bin definitions from ***Interact -> Global Learning -> All Bin Definitions*** page. While adding, or editing a bin definition you can select profile attributes from list of ALL attributes from all mapped profile tables. The types of a Bin Definition can be either Range or List. The “Range” type can only have mathematic operators, the “List” type can only have “contains” operator and consists of list of values.

Example for “Range” type bin:

low income < =30000

30000 < medium income < =60000

high income > 60000

Example for “List” type bin:

New England: MA, NH, CT

North West: MI, IL

A bin definition is global data across all interactive channels and across all learning models.

All bin definitions will be deployed as part of Global Deployment Data. You can deploy them in any interactive channel, deploying once and deployed for ALL. After that, the new bin definitions are saved into a memory cache, which is visible only to the built-in learning sub-system.

When a contact or response event is posted, the value of a profile attribute is mapped to a bin if such bin exists. The "bin" values is used while logging to the learning tables. If bins are defined for the attribute and the attribute value is not part of any bin definitions, then attribute value will be logged as OTHER in learning tables.

Configuring the runtime environment to recognize external learning modules

You can use the Learning Java™ API to write your own learning module. You must configure the runtime environment to recognize your learning utility in Marketing Platform.

You must restart the Interact runtime server for these changes to take effect.

1. In Marketing Platform for the runtime environment, edit the following configuration properties in the `Interact > offerserving` category. The configuration properties for the learning optimizer API exist in `Interact > offerserving > External Learning Config` category.

Configuration property	Setting
`optimizationType`	**ExternalLearning**
`externalLearningClass`	class name for the external learning
`externalLearningClassPath`	The path to the class or JAR files on the runtime server for the external learning. If you are using a server group and all the runtime servers reference the same instance of Marketing Platform, every server must have a copy of the class or JAR files in the same location.

2. Restart the Interact runtime server for these changes to take effect.

Chapter 5. Understanding the Interact API

Interact serves offers dynamically to a wide variety of touchpoints. For example, you can configure the runtime environment and your touchpoint to send messages to your call center employees informing them of the best up sell or cross sell prospects for a customer who has called with a specific type of service inquiry. You can also configure the runtime environment and your touchpoint to provide tailored offers to a customer (visitor) who has entered a particular area of your Web site.

The Interact application programming interface (API) allows you to configure your touchpoint and a runtime server to work together to serve the best possible offers. Using the API, the touchpoint can request information from the runtime server to assign the visitor to a group (a segment) and present offers based on that segment. You can also log data for later analysis to refine your offer presentation strategies.

The Interact API also allows for end-user client to server communication through JavaScript.

In order to provide you with the greatest possible flexibility in integrating Interact with your environments, IBM provides a web service accessible using the Interact API.

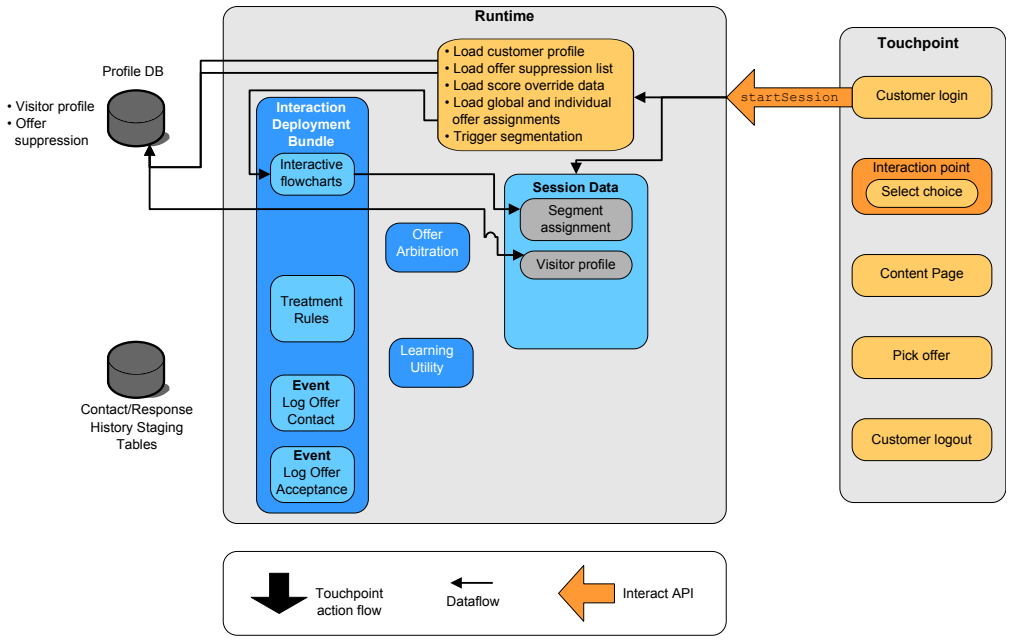
Interact API dataflow

This example shows how the API works between your touchpoint and the runtime environment. The visitor takes only four actions - log in, navigate to page that displays offers, select an offer, and log out. You can design your integration to be as complicated as you need, within the limits of your performance requirements.

This diagram shows a simple implementation of the Interact API.

A visitor logs in to a website and navigates to a page that displays offers. The visitor selects an offer and logs out. While the interaction is simple, several events occur both in the touchpoint and the runtime server:

1. Starting a session
2. Navigating to a page
3. Selecting an offer
4. Closing the session



Starting the session

When the visitor logs in, it triggers a `startSession`.

The `startSession` method does four things:

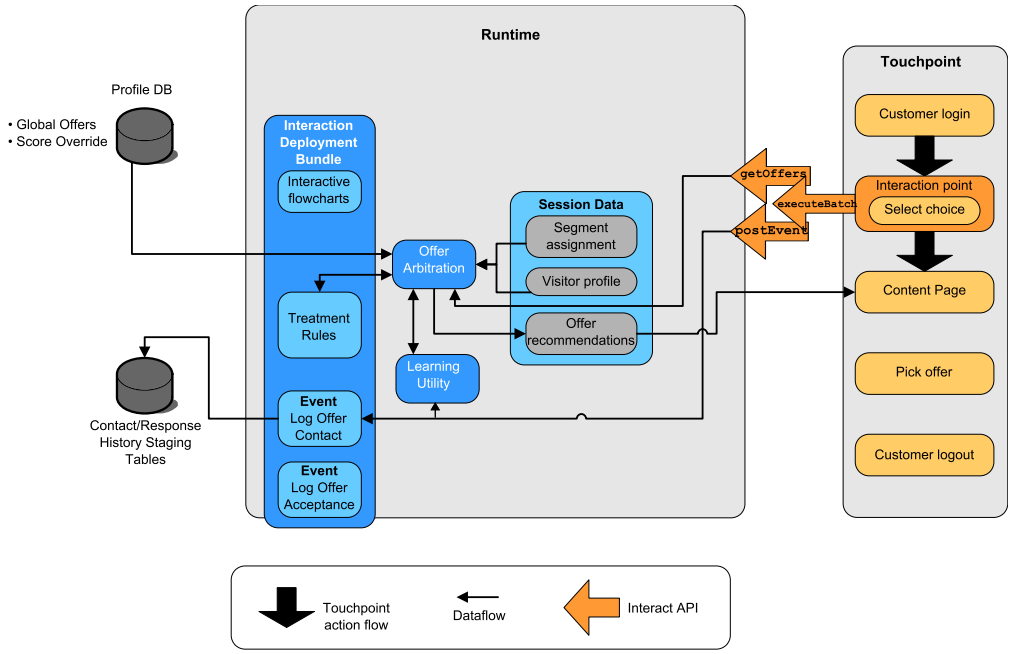
1. It creates a new runtime session
2. It sends a request to load the customer profile data into the session
3. It sends a request to use the profile data and start an interactive flowchart to place the customer into segments. This flowchart run is asynchronous.
4. The runtime server loads any offer suppression and global and individual offer treatment information into the session. The session data is held in memory during the session.

Navigating to a page

The visitor navigates the site until the visitor reaches a pre-defined interaction point. In the figure, the second interaction point (Select choice) is a place where the visitor clicks a link that presents a set of offers. The touchpoint manager configured the link to trigger an `executeBatch` method for selecting an offer.

Selecting an offer

This diagram shows the API call that triggers the `executeBatch` method.

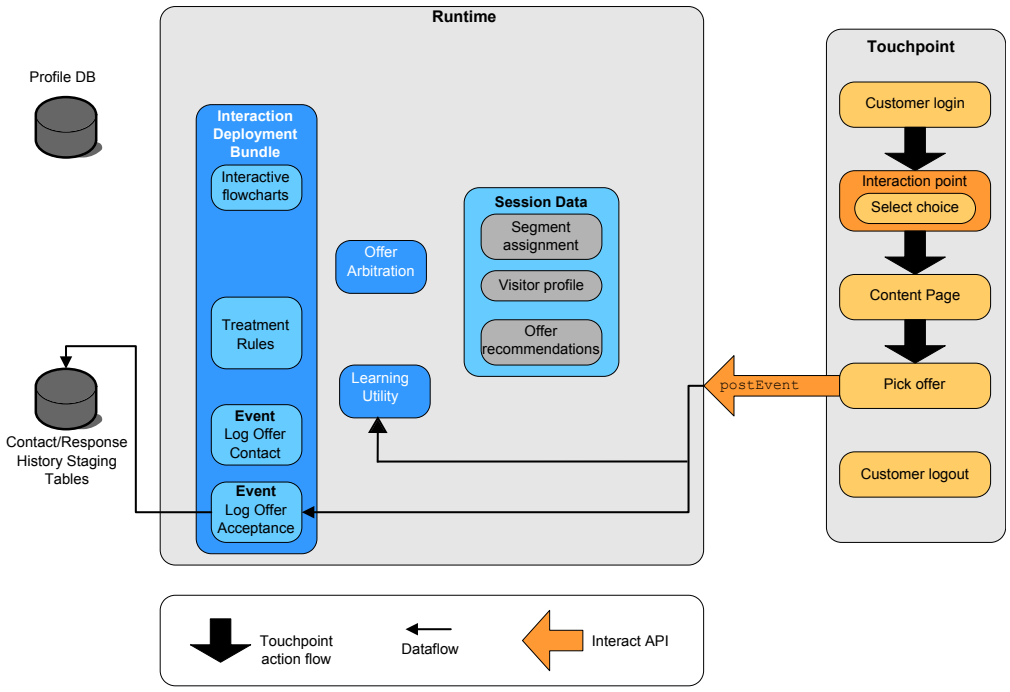


The `executeBatch` method lets you call more than one method in a single call to the runtime server. This particular executeBatch calls two other methods, `getOffers` and `postEvent`. The `getOffers` method requests a list of offers. The runtime server uses the segmentation data, the offer suppression list, the treatment rules, and the learning module to propose a set of offers. The runtime server returns a set of offers that are displayed on the content page.

The `postEvent` method triggers one of the events that are defined in the design environment. In this particular case, the event sends a request to log the offers that are presented to contact history.

The visitor selects one of the offers (Pick offer).

This diagram shows the `postEvent` method.

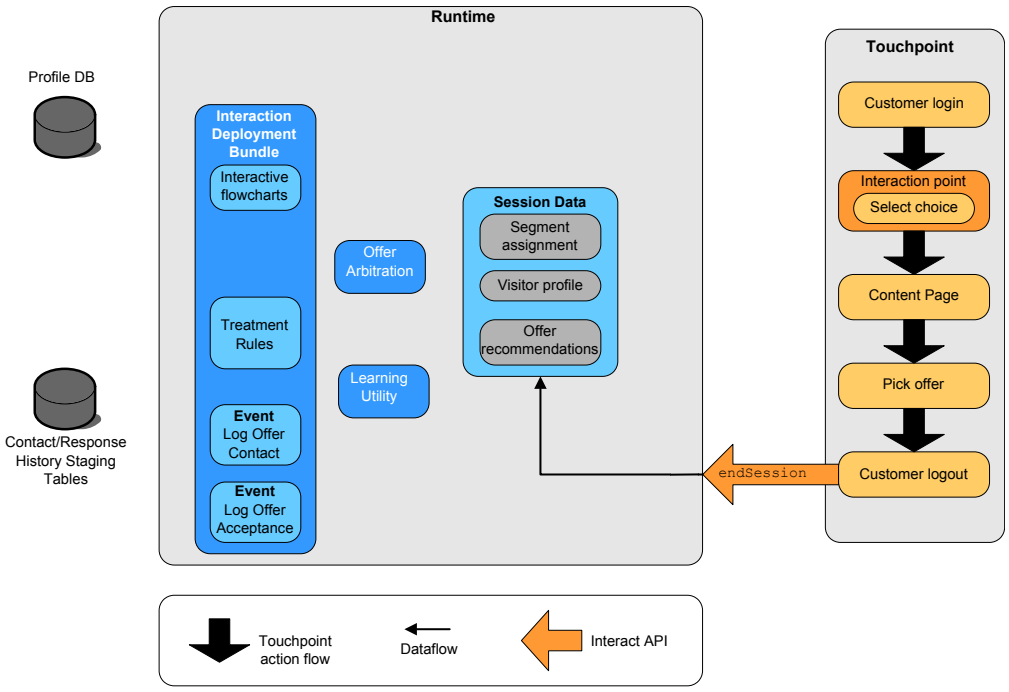


The user interface control that is associated with selecting the offer is configured to send another `postEvent` method. This event sends a request to log the offer acceptance to response history.

Closing the session

After the visitor selects the offer, the visitor is finished with the website and logs out. The log out command is linked to the `endSession` method.

This diagram shows the `endSession` method.



The `endSession` method closes the session. If the visitor forgets to log out, there is a configurable session timeout to ensure that all sessions eventually end. If you want to keep any of the data passed to the session, such as information included in parameters in the `startSession` or `setAudience` methods, work with the person who creates interactive flowcharts. The person who creates an interactive flowchart can use the Snapshot process to write that data to a database before the session ends and that data is lost. You can then use the `postEvent` method to call the interactive flowchart that contains the Snapshot process.

Simple interaction planning example

In this example, you are designing an interaction for a cellular phone company's website. You create three different offers, set up logging for the offers, assign treatment codes to the offer, and show a series of pictures that link to the offers.

Design process

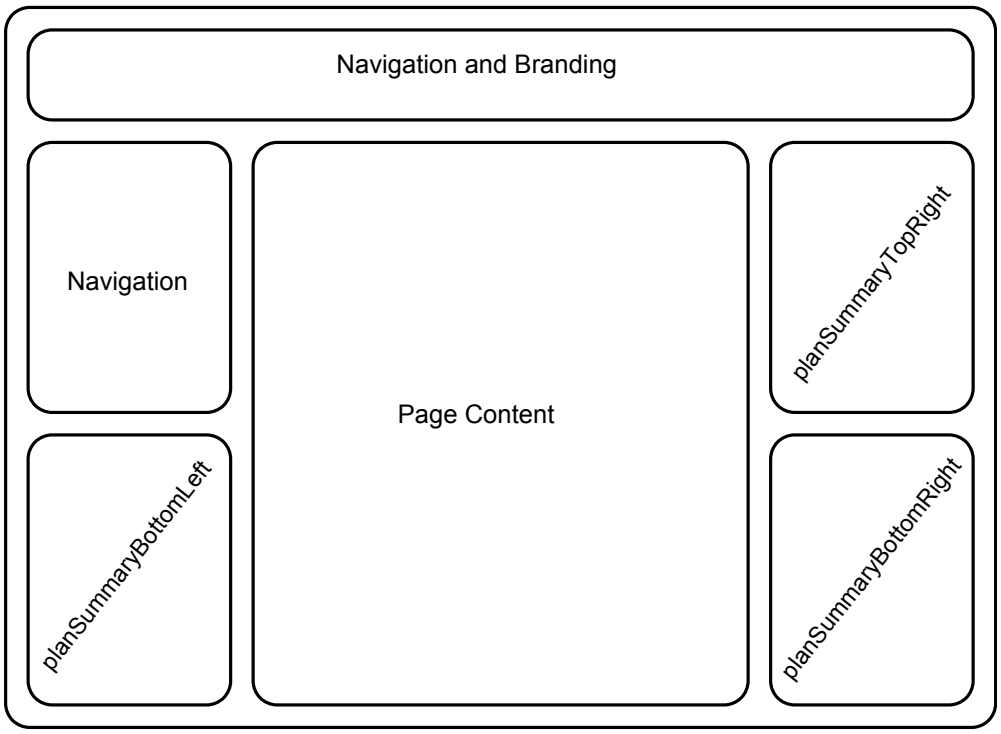
To design an interaction for this client, you:

1. Identify the requirements for the client's summary page
2. Create interaction points for the offer requirements
3. Configure logging for the offers
4. Create treatment codes
5. Link a series of rotating images to the offers

This example is basic, and does not show the best way to write the integration. For example, none of these examples include any error checking that uses the Response class.

Identify requirements for the cell phone plan summary page

The following diagram shows the layout for the cell phone plan summary page.



You define the following items to meet the requirements for the cell phone plan summary page:

Requirement	Implementation
One offer to be displayed in a zone that is dedicated to offers about upgrades The area on the page that displays the upgrade offer must be defined. Also, after Interact picks an offer to display, the information must be logged.	• Interaction point: `ip_planSummaryBottomRight` • Event: `evt_logOffer`
Two offers for phone upgrades Each area on the page that displays the phone upgrades must be defined.	• Interaction point: `ip_planSummaryTopRight` • Interaction point: `ip_planSummaryBottomLeft`
For analysis, you need to log which offers are accepted, and which offers are rejected.	• Event: `evt_offerAccept` • Event: `evt_offerReject`
You also know that you must pass the treatment code of an offer whenever you log an offer contact, acceptance, or rejection.	`NameValuePair`
Display three rotating images on the page. Link the images to the offers.	

Create Interaction points

Now you can ask the design environment user to create the interaction points and events for you while you start to code the integration with your touchpoint.

For each interaction point that displays an offer, you need to first get an offer, then extract the information that you need to display the offer. For example, request an offer for the lower right area of your web page (`planSummaryBottomRight`)

```
Response response=getOffers(sessionID, ip_planSummaryBottomRight, 1)
```

This response call returns a response object that includes an `OfferList` response. However, your web page cannot use an `OfferList` object. You need an image file for the offer, which you know is one of the offer attributes (`offerImg`). You need to extract the offer attribute you need from the `OfferList`.

```
OfferList offerList=response.getOfferList();
if(offerList.getRecommendedOffers() != null)
{
 Offer offer = offerList.getRecommendedOffers()[0];
 NameValuePair[] attributes = offer.getAdditionalAttributes();
 for(NameValuePair attribute: attributes)
 {
  if(attribute.getName().equalsIgnoreCase("offerImg"))
  {
   /* Use this value in your code for the page, for
   example: stringHtml = "<img src="+attribute.getValueAsString()+ " >   */
  }
 }
}
```

Configure logging

Now that you are displaying the offer, you want to log it as a contact.

```
NameValuePair evtParam_TreatmentCode = new NameValuePairImpl();
evtParam_TreatmentCode.setName("UACIOfferTrackingCode");
evtParam_TreatmentCode.setValueAsString(offer.getTreatmentCode());
evtParam_TreatmentCode.setValueDataType(NameValuePair.DATA_TYPE_STRING);
postEvent(sessionID, evt_logOffer, evtParam_TreatmentCode)
```

Instead of calling each of these methods singularly, you can use the `executeBatch` method, as shown in the following example for the `planSummaryBottomLeft` portion of the web page.

```
Command getOffersCommand = new CommandImpl();
getOffersCommand.setMethodIdentifier(Command.COMMAND_GETOFFERS);
getOffersCommand.setInteractionPoint(ip_planSummaryBottomLeft);
getOffersCommand.setNumberRequested(1);

Command postEventCommand = new CommandImpl();
postEventCommand.setMethodIdentifier(Command.COMMAND_POSTEVENT);
postEventCommand.setEvent(evt_logOffer);

/** Build command array */
Command[] commands =
{
 getOffersCommand,
 postEventCommand
};

/** Make the call */
BatchResponse batchResponse = api.executeBatch(sessionId, commands);
```

You do not need to define the UACIOfferTrackingCode in this example. The Interact runtime server automatically logs the last recommended list of treatments as contacts if you do not supply the UACIOfferTrackingCode.

Create treatment codes

Where necessary, you create a NameValuePair to contain the treatment code, as in the following example.

```
NameValuePair evtParam_TreatmentCode = new NameValuePairImpl();
evtParam_TreatmentCode.setName("UACIOfferTrackingCode");
evtParam_TreatmentCode.setValueAsString(offer.getTreatmentCode());
evtParam_TreatmentCode.setValueDataType(NameValuePair.DATA_TYPE_STRING);
```

Link images to offers

For the second area on the page that displays a phone upgrade, you wrote something to change the image displayed every 30 seconds. You decide to rotate between three images and you use the following to retrieve the set of offers to cache for use in your code to rotate the images.

```
Response response=getOffers(sessionID, ip_planSummaryBottomLeft, 3)
OfferList offerList=response.getOfferList();
if(offerList.getRecommendedOffers() != null)
{
 for(int x=0;x<3;x++)
 {
  Offer offer = offerList.getRecommendedOffers()[x];
  if(x==0)
  {
   // grab offerimg attribute value and store somewhere;
   // this will be the first image to display
  }
  else if(x==1)
  {
   // grab offerimg attribute value and store somewhere;
   // this will be the second image to display
  }
  else if(x==2)
  {
   // grab offerimg attribute value and store somewhere;
   // this will be the third image to display
  }
 }
}
```

You must write your client code fetch from the local cache and log to contact only once for each offer after its image is displayed. To log the contact, the UACITrackingCode parameter needs to be posted as before. Each offer has a different tracking code.

```
NameValuePair evtParam_TreatmentCodeSTR = new NameValuePairImpl();
NameValuePair evtParam_TreatmentCodeSBR = new NameValuePairImpl();
NameValuePair evtParam_TreatmentCodeSBL = new NameValuePairImpl();

OfferList offerList=response.getOfferList();
if(offerList.getRecommendedOffers() != null)
 {
 for(int x=0;x<3;x++)
  {
  Offer offer = offerList.getRecommendedOffers()[x];
  if(x==0)
  {
   evtParam_TreatmentCodeSTR.setName("UACIOfferTrackingCode");
   evtParam_TreatmentCodeSTR.setValueAsString(offer.getTreatmentCode());
```

```
   evtParam_TreatmentCodeSTR.setValueDataType(NameValuePair.DATA_TYPE_STRING);
  }
  else if(x==1)
  {
   evtParam_TreatmentCodeSBR.setName("UACIOfferTrackingCode");
   evtParam_TreatmentCodeSBR.setValueAsString(offer.getTreatmentCode());
   evtParam_TreatmentCodeSBR.setValueDataType(NameValuePair.DATA_TYPE_STRING);
  }
  else if(x==2)
  {
   evtParam_TreatmentCodeSBL.setName("UACIOfferTrackingCode");
   evtParam_TreatmentCodeSBL.setValueAsString(offer.getTreatmentCode());
   evtParam_TreatmentCodeSBL.setValueDataType(NameValuePair.DATA_TYPE_STRING);
  }
 }
}
```

For each offer, if the offer is clicked, you log the offer that is accepted and the offers that are rejected. (In this scenario, offers not explicitly selected are considered rejected.) The following is an example if the `ip_planSummaryTopRight` offer is selected:

```
postEvent(sessionID, evt_offerAccept, evtParam_TreatmentCodeSTR)
postEvent(sessionID, evt_offerReject, evtParam_TreatmentCodeSBR)
postEvent(sessionID, evt_offerReject, evtParam_TreatmentCodeSBL)
```

In practice, it would be best to send these three `postEvent` calls with the `executeBatch` method.

Designing the Interact API integration

Building your Interact API integration with your touchpoint requires some designing before you can begin implementation. You need to work with your marketing team to decide on where in your touchpoint you want the runtime environment to serve offers (define your interaction points) and what other kind of tracking or interactive functionality you want to use (define your events).

In the design phase, these may be mere outlines. For example, for a telecommunications web site, the customer's plan summary page should display one offer regarding plan upgrade and two offers for phone upgrades.

Once your company has decided where and how they wish to interact with customers, you need to use Interact to define the details. A flowchart author needs to design the interactive flowcharts that will be used when re-segmentation events occur. You need to decide on the number and names of interaction points and events, as well as what data needs to be passed along for proper segmentation, event posting, and offer retrieval. The design environment user defines the interaction points and events for the Interactive Channel. You then use those names as you code the integration with your touchpoint in the runtime environment. You should also define what metric information is required, to define when you need to log offer contacts and responses.

Points to consider

When you design an interaction, keep in mind the effects that no eligible offer, an unreachable runtime server, process timing have on the interaction. Be specific when you define offer rejections. Consider the optional product features that can enhance the interaction.

When you are designing your interaction:

Create some default filler content

Create default filler content, a benign branding message or empty content, for every interaction point where offers can be presented. This filler content is used when there are no offers eligible to be served to the current visitor in the current situation. You assign this default filler content as the default string for the interaction point.

Include an alternative method of presenting content

Include some method of presenting content in case your touchpoint cannot reach the runtime server group for some unforeseen reason.

Consider the time that running flowcharts takes

When you trigger events that resegment your visitor, including `postEvent` and `setAudience`, keep in mind that running flowcharts does take some amount of time. The `getOffers` method waits until segmentation is finished before the `getOffers` method runs. Overly frequent resegmentation can hinder `getOffers` call response performance.

Decide what an "offer rejection" means

Several reports, such as the Channel Offer Performance Summary report, present the number of times an offer is rejected. This report shows the number of times a `postEvent` triggered a Log Offer Rejection action. You need to determine whether the Log Offer Rejection action is for an actual rejection, such as clicking a link labeled **No, thanks**. Or is Log Offer Rejection action for an offer that is ignored, such as a page that displays three different banner ads, none of which are selected.

Decide which offer selection features to use

There are several optional features you can use to enhance Interact offer selection. These features include:

- Learning
- Offer suppression
- Individual offer assignments
- Other elements of offer serving

You need to determine how many, if any, of these optional features would enhance your interactions.

Chapter 6. Managing the IBM Interact API

Whenever you use the `startSession` method, you create a Interact runtime session on the runtime server. You can use configuration properties to manage the sessions on a runtime server.

You may need to configure these settings as you implement your Interact integration with your touchpoint.

These configuration properties are in the `sessionManagement` category.

Locale and the Interact API

You can use Interact for non-English touchpoints. The touchpoint and all strings in the API use the locale defined for the runtime environment user.

You can select only one locale per server group.

For example, in the runtime environment, you create two users, asm_admin_en with the user locale set to English, and asm_admin_fr with the user locale set to French. If your touchpoint is designed for French speakers, define the `asmUserForDefaultLocale` property for the runtime environment as asm_admin_fr.

About JMX monitoring

Interact provides Java Management Extensions (JMX) monitoring service that you can access with any JMX monitoring application. This JMX monitoring enables you to monitor and manage your runtime servers.

The JMX attributes provide a lot of detailed information about the runtime server. For example, the JMX attribute `ErrorCount` gives the number of error messages logged since last reset or system start. You can use this information to see how often there are errors in your system. If you have coded your web site to only call an end session if someone completes a transaction, you could also compare the `startSessionCount` to the `endSessionCount` to see how many transactions are incomplete.

Interact supports the RMI and JMXMP protocols, as defined by JSR 160. You can connect to the JMX monitoring service with any JSR160-compliant JMX client.

Interactive flowcharts can be monitored with JMX monitoring only. Information about Interactive flowcharts does not appear in Campaign Monitoring.

Note: If you are using IBM WebSphere® with a node manager, you must define the Generic JVM Argument to enable JMX monitoring.

Configuring Interact to use JMX monitoring with the RMI protocol

Use this procedure to configure Interact to use JMX monitoring with the RMI protocol.

The default address for monitoring for the RMI protocol is `service:jmx:rmi:///jndi/rmi://RuntimeServer:port/interact`.

In Marketing Platform for the runtime environment, edit the following configuration properties in the `Interact > monitoring` category.

Configuration property	Setting
`protocol`	**RMI**
`port`	The port number for the JMX service
`enableSecurity`	**False** The Interact implementation of the RMI protocol does not support security.

Configuring Interact to use JMX monitoring with the JMXMP protocol

Use this procedure to configure Interact to use JMX monitoring with the JMXMP protocol.

The JMXMP protocol requires two extra libraries in the following order in the classpath, `InteractJMX.jar` and `jmxremote_optional.jar`. Both of these files can be found in the `lib` directory of your runtime environment installation.

If you enable security, the user name and password must match a user in Marketing Platform for the runtime environment. You cannot use an empty password.

The default address for monitoring for the JMXMP protocol is `service:jmx:jmxmp://RuntimeServer:port`.

1. Verify that the `InteractJMX.jar` and `jmxremote_optional.jar` libraries are in the classpath in order. If they are not in the classpath, add them to the classpath.
2. In Marketing Platform for the runtime environment, edit the following configuration properties in the `Interact > monitoring` category.

Configuration property	Setting
`protocol`	**JMXMP**
`port`	the port number for the JMX service
`enableSecurity`	**False** to disable security, or **True** to enable security

Configuring Interact to use the jconsole scripts for JMX monitoring

If you do not have a separate JMX monitoring application, you can use the jconsole that is installed with the JVM. You can start the jconsole with the startup scripts in the `Interact/tools` directory.

The jconsole script uses the JMXMP protocol for monitoring by default. The default settings for `jconsole.bat` are:

The JMXMP connection

```
%JAVA_HOME%\bin\jconsole.exe -J-Djava.class.path=%JAVA_HOME%
  \lib\jconsole.jar;INTERACT_LIB%\interactJMX.jar; INTERACT_LIB%
  \jmxremote_optional.jar service:jmx:jmxmp://%HOST%:%PORT%
```

The RMI connection

```
%JAVA_HOME%\bin\jconsole.exe -J-Djava.class.path=%JAVA_HOME%
      \lib\jconsole.jar;INTERACT_LIB%\jmxremote_optional.jar
     service:jmx:rmi:///jndi/rmi://%HOST%:%PORT%/interact
```

1. Open `Interact\tools\jconsole.bat` (Windows) or `Interact/tools/jconsole.sh` (UNIX) in a text editor.
2. Set `INTERACT_LIB` to the full path to the *`InteractInstallationDirectory`*`/lib` directory.
3. Set `HOST` to the host name of the runtime server you want to monitor.
4. Set `PORT` to the port you configured JMX to listen on with the `Interact > monitoring > port` property.
5. Optional: If you are using the RMI protocol for monitoring, add a comment before the JMXMP connection and remove the comment before the RMI connection.

JMX attributes

There are multiple attributes available for JMX monitoring. Design environment attributes include contact response history ETL monitoring. Runtime environment attributes include exceptions, several different flowchart attributes, locale, logger, and thread pool statistics. Several service statistics attributes are also available. All data that is provided by JMX monitoring is since the last reset or system start. For example, a count is of the number of items since last reset or system start, not since installation.

Contact Response History ETL Monitor attributes

The Contact Response History ETL Monitor attributes are part of the design environment. All of the following attributes are part of the runtime environment.

Table 9. Contact Response History ETL Monitor

Attribute	Description
AvgCHExecutionTime	The average number of milliseconds it takes for the contact and response history module to write to the contact history table. This average is calculated only for the operations that were successful and for which there was at least one record that was written to the contact history table.
AvgETLExecutionTime	The average number of milliseconds it takes for the contact and response history module to read data from the runtime environment. The average includes the time for successful as well as failed operations.
AvgRHExecutionTime	The average number of milliseconds it takes for the contact and response history module to write to the response history table. This average is calculated only for the operations that were successful and for which there was at least one record that was written to the response history table.

Table 9. Contact Response History ETL Monitor (continued)

Attribute	Description
ErrorCount	The number of error messages that were logged since last reset or system start, if any.
HighWaterMarkCHExecutionTime	The maximum number of milliseconds it took for the contact and response history module to write to the contact history table. This value is calculated only for the operations that were successful and for which there was at least one record that was written to the contact history table.
HighWaterMarkETLExecutionTime	The maximum number of milliseconds it took for the contact and response history module to read data from the runtime environment. The calculation includes both successful as well as failed operations.
HighWaterMarkRHExecutionTime	The maximum number of milliseconds it took for the contact and response history module to write to the response history table. This value is calculated only for the operations that were successful and for which there was at least one record that was written to the response history table.
LastExecutionDuration	The number of milliseconds the contact and response history module took to perform the last copy.
NumberOfExecutions	The number of times the contact and response history module has run since initialization.
LastExecutionStart	The time the last run of the contact and response history module started.
LastExecutionSuccessful	If true, the last run of the contact and response history module was successful. If false, an error occurred.
NumberOfContactHistoryRecordsMarked	The number of contact history records in the `UACI_CHStaging` table that are being moved during the current run of the contact and response history module. This value is greater than zero only if the contact and response history module is running.
NumberOfResponseHistoryRecordsMarked	The number of response history records in the `UACI_RHStaging` table that are being moved during the current run of the contact and response history module. This value is greater than zero only if the contact and response history module is running.

Exception attributes

Exception attributes are part of the runtime environment.

Table 10. Exceptions

Attribute	Description
errorCount	The number of error messages that were logged since last reset or system start.
warningCount	The number of warning messages that were logged since last reset or system start.

Flowchart Engine Statistics attributes

Flowchart Engine Statistics attributes are part of the runtime environment.

Table 11. Flowchart Engine Statistics

Attribute	Description
activeProcessBoxThreads	Active count of flowchart process threads (shared between all executions) that are currently running.
activeSchedulerThreads	Active count of Flowchart Scheduler threads that are currently running.
avgExecutionTimeMillis	Average flowchart execution time in milliseconds.
CurrentJobsInProcessBoxQueue	The number of jobs that are waiting to be run by flowchart process threads.
CurrentJobsInSchedulerQueue	The number of jobs that are waiting to be run by Flowchart Scheduler threads.
maximumProcessBoxThreads	Maximum number of flowchart process threads (shared between all executions) that can be run.
maximumSchedulerThreads	Maximum number of Flowchart Scheduler threads (one thread per execution) that can be run.
numExecutionsCompleted	Total number of flowchart executions that completed.
numExecutionsStarted	Total number of flowchart executions started.

Specific flowcharts by interactive channel attributes

Specific flowcharts by interactive channel attributes are part of the runtime environment.

Table 12. Specific flowcharts by interactive channel

Attribute	Description
AvgExecutionTimeMillis	Average execution time in milliseconds for this flowchart in this interactive channel.

Table 12. Specific flowcharts by interactive channel (continued)

Attribute	Description
HighWaterMarkForExecutionTime	Maximum execution time in milliseconds for this flowchart in this interactive channel.
LastCompletedExecutionTimeMillis	Execution time in milliseconds for the last completion of this flowchart in this interactive channel.
NumExecutionsCompleted	Total number of executions that have completed for this flowchart in this interactive channel.
NumExecutionsStarted	Total number of executions that are started for this flowchart in this interactive channel.

Locale attributes

Locale attributes are part of the runtime environment.

Table 13. Locale

Attribute	Description
locale	Locale setting for JMX client.

Logger Configuration attributes

Logger Configuration attributes are part of the runtime environment.

Table 14. Logger Configuration

Attribute	Description
category	Change the log category on which the log level can be manipulated.

Services Thread Pool Statistics attributes

Services Thread Pool Statistics attributes are part of the runtime environment.

Table 15. Services Thread Pool Statistics

Attribute	Description
activeContactHistThreads	The approximate number of threads that are actively running tasks for Contact History and Response History.
activeFlushCacheToDBThreads	The approximate number of threads that are actively running tasks to flush cached statistics to the data store.
activeOtherStatsThreads	The approximate number of threads that are actively running tasks for Eligible Stats, Event Activities, and Default Stats.
CurrentHighWaterMarkInContactHistQueue	Greatest number of entries queued to be logged by the service that collects the contact and response history data.

Table 15. Services Thread Pool Statistics (continued)

Attribute	Description
CurrentHighWaterMark InFlushCachetoDBQueue	Greatest number of entries queued to be logged by the service that writes the data in the cache to the database tables.
CurrentHighWaterMarkInOtherStatsQueue	Greatest number of entries queued to be logged by the service that collects the offer eligibility statistics, default string usage statistics, event activity statistics, and the custom log to table data.
currentMsgsInContactHistQueue	The number of jobs in the queue for the thread pool that is used for Contact History and Response History.
currentMsgsInFlushCacheToDBQueue	The number of jobs in the queue for the thread pool that is used to flush cached statistics to the data store.
currentMsgsInOtherStatsQueue	The number of jobs in the queue for the thread pool that is used for Eligible Stats, Event Activities, and Default Stats.
maximumContactHistThreads	The largest number of threads that have ever simultaneously been in the pool that is used for Contact History and Response History.
maximumFlushCacheToDBThreads	The largest number of threads that have ever simultaneously been in the pool that is used for flushing cached statistics to the data store.
maximumOtherStatsThreads	The largest number of threads that have ever simultaneously been in the pool that is used for Eligible Stats, Event Activities, and Default Stats.

Service Statistics attributes

The Service Statistics consist of a set of attributes for each service.

- ContactHistoryMemoryCacheStatistics - The service that collects data for the contact history staging tables.
- CustomLoggerStatistics - The service that collects custom data to write to a table (an event that uses the `UACICustomLoggerTableName` event parameter).
- Default Statistics - The service that collects the statistics regarding the number of times the default string for the interaction point was used.
- Eligibility Statistics - The service that writes the statistics for eligible offers.
- Event Activity Statistics - The service that collects the event statistics, both system events such as `getOffer` or `startSession` and user events that are triggered by `postEvent`.
- Response History Memory Cache Statistics - The service that writes to the response history staging tables.
- Cross-session Response Statistics - The service that collects the cross-session response tracking data.

Table 16. Service Statistics

Attribute	Description
Count	The number of messages processed.
ExecTimeInsideMutex	The amount of time spent processing messages for this service, excluding time spent waiting for other threads, in milliseconds. If there is a great difference between ExecTimeInsidMutex and ExecTimeMillis, you might need to change the thread pool size for the service.
ExecTimeMillis	The amount of time spent processing messages for this service, including time spent waiting for other threads, in milliseconds.
ExecTimeOfDBInsertOnly	The amount of time in milliseconds spent processing the batch insert portion only.
HighWaterMark	The maximum number of messages that are processed for this service.
NumberOfDBInserts	The total number of batch inserts run.
TotalRowsInserted	The total number of rows that are inserted into the database.

Service Statistics - Database Load Utility attributes

Service Statistics - Database Load Utility attributes are part of the runtime environment.

Table 17. Service Statistics - Database Load Utility

Attribute	Description
ExecTimeOfWriteToCache	The amount of time in milliseconds spent writing to file cache, including writing to files and getting the primary key from database when necessary.
ExecTimeOfLoaderDBAccessOnly	The amount of time in milliseconds spent running database loader portion only.
ExecTimeOfLoaderThreads	The amount of time in milliseconds spent by database loader threads.
ExecTimeOfFlushCacheFiles	The amount of time in milliseconds spent flushing the cache and re-creating new ones.
ExecTimeOfRetrievePKDBAccess	The amount of time in milliseconds spent retrieving the primary key database access.
NumberOfDBLoaderRuns	The total number of database loader runs.
NumberOfLoaderStagingDirCreated	The total number of staging directories that are created.
NumberOfLoaderStagingDirRemoved	The total number of staging directories that are removed.
NumberOfLoaderStaging DirMovedToAttention	The total number of staging directories that are renamed to attention.

Table 17. Service Statistics - Database Load Utility (continued)

Attribute	Description
NumberOfLoaderStagingDirMovedToError	The total number of staging directories that are renamed to error.
NumberOfLoaderStagingDirRecovered	The total number of staging directories recovered, including at startup time and rerun by background threads.
NumberOfTimesRetrievePKFromDB	The total number of times the primary key was retrieved from database.
NumberOfLoaderThreadsRuns	The total number of database loader threads runs.
NumberOfFlushCacheFiles	The total number of times the file cache was flushed.

API Statistics attributes

API Statistics attributes are part of the runtime environment.

Table 18. API Statistics

Attribute	Description
endSessionCount	The number of `endSession` API calls since last reset or system start.
endSessionDuration	Time that is elapsed for the last `endSession` API call in milliseconds.
executeBatchCount	The number of `executeBatch` API calls since last reset or system start.
executeBatchDuration	Time that is elapsed for the last `executeBatch` API call in milliseconds.
getOffersCount	The number of `getOffers` API calls since last reset or system start.
getOffersDuration	Time that is elapsed for the last `getOffer` API call in milliseconds.
getProfileCount	The number of `getProfile` API calls since last reset or system start.
getProfileDuration	Time that is elapsed for the last `getProfileDuration` API call in milliseconds.
getVersionCount	The number of `getVersion` API calls since last reset or system start.
getVersionDuration	Time that is elapsed for the last `getVersion` API call in milliseconds.
loadOfferSuppressionDuration	Time that is elapsed for the last `loadOfferSuppression` API call.
LoadOffersBySQLCount	The number of `LoadOffersBySQL` API calls since last reset or system start.
LoadOffersBySQLDuration	Time that is elapsed for the last `LoadOffersBySQL` API call in milliseconds.
loadProfileDuration	Time that is elapsed for the last `loadProfile` API call in milliseconds.

Table 18. API Statistics (continued)

Attribute	Description
loadScoreOverrideDuration	Time that is elapsed for the last `loadScoreOverride` API call in milliseconds.
postEventCount	The number of `postEvent` API calls since last reset or system start.
postEventDuration	Time that is elapsed for the last `postEvent` API call in milliseconds.
runSegmentationDuration	Time that is elapsed for the last `runSegmentation` API call in milliseconds.
setAudienceCount	The number of `setAudience` API calls since last reset or system start.
setAudienceDuration	Time that is elapsed for the last `setAudience` API call in milliseconds.
setDebugCount	The number of `setDebug` API calls since last reset or system start.
setDebugDuration	Time that is elapsed for the last `setDebug` API call in milliseconds.
startSessionCount	The number of `startSession` API calls since last reset or system start.
startSessionAverage	Average time that is elapsed for the last `startSession` API call in milliseconds.
ActiveSessionCount	The number of sessions that are currently active in the interact run time instance. **Note:** The ActiveSessionCount in JMX MBean com.unicacorp.interact:type=api, group=Statistics does not consider timed out events and hence it could show incorrect active count.

Learning Optimizer Statistics attributes

Learning Optimizer Statistics attributes are part of the runtime environment.

Table 19. Learning Optimizer Statistics

Attribute	Description
LearningOptimizerAcceptCalls	The number of accept events that are passed into the learning module.
LearningOptimizer AcceptTrackingDuration	The total number of milliseconds spent logging the accept events in the learning module.
LearningOptimizerContactCalls	The number of contact events that are passed into the learning module.
LearningOptimizer ContactTrackingDuration	The total number of milliseconds spent logging the contact events in the learning module.
LearningOptimizerLogOtherCalls	The number of non-contact and non-accept events that are passed into the learning module.

Table 19. Learning Optimizer Statistics (continued)

Attribute	Description
LearningOptimizer LogOtherTrackingDuration	The duration in milliseconds spent in logging other events (non-contact and non-accept) in the learning module.
LearningOptimizer NonRandomCalls	The number of times the configured learning implementation was applied.
LearningOptimizer RandomCalls	The number of times the configured learning implementation was bypassed and random selection was applied.
LearningOptimizer RecommendCalls	The number of recommend requests that are passed into the learning module.
LearningOptimizer RecommendDuration	The total number of milliseconds spent in the learning recommend logic.

Default Offer Statistics attributes

Default Offer Statistics attributes are part of the runtime environment.

Table 20. Default Offer Statistics

Attribute	Description
LoadDefaultOffersDuration	Time that is elapsed on the default offers loading.
DefaultOffersCalls	The number of times the default offers loading.

Triggered Message Dispatchers attributes

Triggered Message Dispatchers attributes are part of the runtime environment.

Table 21. Triggered Message Dispatchers

Attribute	Description
NumRequested	The total number of offers that were requested for dispatching using this dispatcher.
NumDispatched	The total number of offers this dispatcher successfully dispatched.
AvgExecutionTime	The average time in milliseconds this dispatcher uses for dispatching an offer. Only the offers that were successfully dispatched to gateways are counted in the calculation.
CurrentQueueSize	The number of offers currently waiting to be dispatched.
GatewayInvocation	The number of offers and average dispatching time in milliseconds dispatched to each gateway by this dispatcher. The format of its value is `{gateway name=[number of offers, average dispatching time]}`.

Triggered Message Gateways attributes

Triggered Message Gateways attributes are part of the runtime environment.

Table 22. Triggered Message Gateways

Attribute	Description
NumValidationRequested	The total number of offers this gateway requested for validation.
NumValidated	The total number of offers this gateway successfully validated.
AvgValidationTime	The average time in milliseconds this gateway uses for validating an offer. Only the offers that were successfully validated are counted in the calculation.
NumDeliveryRequested	The total number of offers this gateway requested for delivery.
NumDelivered	The total number of offers this gateway successfully delivered.
AvgDeliveryTime	The average time in milliseconds this gateway uses for delivering an offer. Only the offers that were successfully delivered are counted in the calculation.

Triggered Message Messages attributes

Triggered Message Messages attributes are part of the runtime environment.

Table 23. Triggered Message Messages

Attribute	Description
ProcessSuccessCount	The total number of times this triggered message successfully executed.
AvgSuccessProcessTime	The average time in milliseconds this triggered message spends for each successful execution.
ProcessErrorCount	The total number of times this triggered message unsuccessfully executed.
AvgErrorProcessTime	The average time in milliseconds this triggered message spends for each unsuccessful execution.
SelectBranchCount	The total number of times branch selection was executed while processing triggered messages.
AvgSelectBranchTime	The average time in milliseconds branch selection execution uses while processing triggered messages.
SelectOfferCount	The total number of times offer selection was executed while processing triggered messages.

Table 23. Triggered Message Messages (continued)

Attribute	Description
AvgSelectOfferTime	The average time in milliseconds offer selection execution uses while processing triggered messages.
SelectChannelCount	The total number of times channel selection was executed while processing triggered messages.
AvgSelectChannelTime	The average time in milliseconds channel selection execution uses while processing triggered messages.
FlowchartWaitCount	The total number of times this triggered message waited for segmentation to complete.
AvgFlowchartWaitTime	The average time in milliseconds this triggered message waited for segmentation to complete in each execution.
WaitFlowchartTimeoutCount	The total number of times this triggered message timed out while waiting for segmentation to complete.

JMX operations

There are several operations available for JMX monitoring.

The following table describes the operations available for JMX monitoring.

Group	Attribute	Description
Logger Configuration	activateDebug	Set log level for the log file that is defined in `Interact/conf/interact_log4j.properties` to debug.
Logger Configuration	activateError	Set log level for the log file that is defined in `Interact/conf/interact_log4j.properties` to error.
Logger Configuration	activateFatal	Set log level for the log file that is defined in `Interact/conf/interact_log4j.properties` to fatal.
Logger Configuration	activateInfo	Set log level for the log file that is defined in `Interact/conf/interact_log4j.properties` to info.
Logger Configuration	activateTrace	Set log level for the log file that is defined in `Interact/conf/interact_log4j.properties` to trace.
Logger Configuration	activateWarn	Set log level for the log file that is defined in `Interact/conf/interact_log4j.properties` to warn.
Locale	changeLocale	Change the JMX client's locale. Interact supported locales are `de`, `en`, `es`, and `fr`.
ContactResponseHistory ETLMonitor	reset	Reset all counters.

Group	Attribute	Description
Default Offer Statistics	updatePollPeriod	Updates defaultOfferUpdatePollPeriod. This value, in seconds, tells the system how long to wait before the system updates the default offers in the cache. If set to -1, the system reads the number of default offers only at startup.

Chapter 7. Classes and methods for the IBM Interact Java, SOAP, and REST API

The following sections list requirements and other details you should know before you work with the Interact API.

Note: This section assumes you are familiar with your touchpoint, the Java programming language, and working with a Java-based API.

The Interact API has a Java client adaptor that uses Java serialization over HTTP. In addition, Interact supplies a WSDL to support SOAP clients. The WSDL exposes the same set of functions as the Java client adaptor, so the following sections, except for examples, still apply.

Note: Multiple occurrences of any parameter in a single API call is not supported.

Interact API Classes

The Interact API is based on the `InteractAPI` class.

There are 6 supporting interfaces.

- `AdvisoryMessage`
- `BatchResponse`
- `NameValuePair`
- `Offer`
- `OfferList`
- `Response`

These interfaces have 3 supporting concrete classes. The following two concrete classes need to be instantiated and passed in as arguments into the Interact API methods:

- NameValuePairImpl
- CommandImpl

A third concrete class, called `AdvisoryMessageCode` is available to provide the constants used to distinguish the message codes returned from the server whenever applicable.

The rest of this section describes the methods which comprise the Interact API.

Java serialization over HTTP prerequisites

The Java client adapter uses Java serialization over HTTP.

The prerequisites for using the Java client adapter for Java serialization over HTTP are:

1. Add the following file to your `CLASSPATH`:

   `Interact_Home/lib/interact_client.jar`
2. All objects that are passed back and forth between the client and the server can be found in the package `com.unicacorp.interact.api`. For more details, see the

Interact API Javadoc installed on the runtime server in `Interact_Home/docs/apiJavaDoc`. You can view the Javadoc by opening the `index.html` file in that location with any web browser.

3. To get an instance of the `InteractAPI` class, call the static method `getInstance` with the url of the Interact runtime server.

SOAP prerequisites

Before you can access the runtime server with SOAP, you do several prerequisite tasks to configure your environment.

Important: Performance testing shows that the Java serialization adapter performs at a much higher rate than a generated SOAP client. For best performance, use the Java serialization adapter whenever possible.

To access the runtime server with SOAP, you must do the following:

1. Convert the Interact API WSDL with the SOAP toolkit of your choice.

   The Interact API WSDL is installed with Interact in the `Interact/conf` directory.

   When you configure SOAP using the WSDL XML files, you must modify your URLs to the host name and port of the runtime server.

   The text of the WSDL is available at the end of Interact Administration guide.
2. Install and configure the runtime server.

   The runtime server must be running to fully test your integration.
3. Verify that you are using the correct SOAP version.

   Interact uses axis2 1.3 as the SOAP infrastructure on the Interact runtime servers. For details about what versions of SOAP axis2 1.3 supports, see the following website:

   Apache Axis2

   Interact was tested with the axis2, XFire, JAX-WS-Ri, DotNet, SOAPUI, and IBM RAD SOAP clients.

REST prerequisites

One method of calling the Interact API is by using JSON (JavaScript Object Notation) format calls over HTTP, referred to here as the REST API. The REST API has the advantage of having better performance than SOAP, although the Java serialization adapter is still the fastest method for Interact API calls.

Before you begin using the REST API, be aware of the following:

- The URL that supports REST calls to the Interact API is:

  `http://Interact_Runtime_Server:PORT/interact/servlet/RestServlet`, substituting the actual host name or IP address of the Interact runtime server and the port on which Interact is deployed.
- There are two Interact classes specific to the REST API: `RestClientConnector`, which serves as a helper to connect to an Interact run time instance via REST with the format of JSON, and `RestFieldConstants`, which describes the underlying format of the JSON message that is used for API requests and responses.
- A sample REST client is provided at `Interact _Home/samples/javaApi/InteractRestClient.java`. Although the sample code is a simple example, it should provide a good starting point for demonstrating how the REST API is used.

- For a complete description of the REST API classes along with all other Interact API information, see the Javadoc installed on the runtime server at `Interact_Home/docs/apiJavaDoc`.
- The REST API returns SessionIDs and messages in the HTML-escaped format and not in the Unicode format.

Other than the information mentioned here, the REST API supports all of the methods that are supported by the other protocols for using the Interact API.

API JavaDoc

In addition to Interact Administrator guide, the Javadoc for the Interact API is installed with the runtime server. The Javadoc is installed for your reference in the `Interact_Home/docs/apiJavaDoc` directory.

API examples

All of the examples in the guide were created with the Java serialization over HTTP adapter. The classes generated from the WSDL can vary based on the SOAP toolkit and the options you select. If you are using SOAP, these examples might not work the same in your environment.

Working with session data

When you initiate a session with the `startSession` method, session data is loaded into memory. Throughout the session, you can read and write to the session data (which is a superset of the static profile data).

The session contains the following data:

- Static profile data
- Segment assignments
- Real-time data
- Offer recommendations

All session data is available until you call the `endSession` method, or the `sessionTimeout` time elapses. Once the session ends, all data not explicitly saved to contact or response history or some other database table is lost.

The data is stored as a set of name-value pairs. If the data is read from a database table, the name is the column of the table.

You can create these name-value pairs as you work with the Interact API. You do not need to declare all name-value pairs in a global area. If you set new event parameters as name-value pairs, the runtime environment adds the name-value pairs to the session data. For example if you use event parameters with the `postEvent` method, the runtime environment adds the event parameters to the session data, even if the event parameters were not available in the profile data. This data exists in the session data only.

You can overwrite session data at any time. For example, if part of the customer profile includes `creditScore`, you can pass in an event parameter using the custom type `NameValuePair`. In the `NameValuePair` class, you can use the `setName` and `setValueAsNumeric` methods to change the value. The name needs to match. Within the session data, the name is not case-sensitive. Therefore, the name `creditscore` or `CrEdItScOrE` would both overwrite `creditScore`.

Only the last data written to the session data is kept. For example, `startSession` loads the profile data for the value of `lastOffer`. A `postEvent` method overwrites `lastOffer`. Then a second `postEvent` method overwrites `lastOffer`. The runtime environment keeps only the data written by the second `postEvent` method in the session data.

When the session ends, the data is lost, unless you made special considerations such as using a Snapshot process in your interactive flowchart to write the data to a database table. If you are planning on using Snapshot processes, remember that the names need to match the limitations of your database. For example, if your are allowed only 256 characters for the name of a column, then the name for the name-value pair should not exceed 256 characters.

About the InteractAPI class

The InteractAPI class contains the methods which you use to integrate your touchpoint with the runtime server. All other classes and methods in the Interact API support the methods in this class.

You must compile your implementation against `interact_client.jar` located in the `lib` directory of your Interact runtime environment installation.

endSession

The `endSession` method marks the end of the runtime session. When the runtime server receives this method, the runtime server logs to history, clears memory, and so on.

`endSession(String sessionID)`

- **sessionID** - Unique string identifying the session.

If the endSession method is not called, runtime sessions timeout. The timeout period is configurable with the `sessionTimeout` property.

Return value

The runtime server responds to the `endSession` method with the `Response` object with the following attributes populated:

- SessionID
- ApiVersion
- StatusCode
- AdvisoryMessages

Example

The following example shows the `endSession` method and how you can parse the response. `sessionId` is the same string to identify the session used by the `startSession` call which started this session.

```
response = api.endSession(sessionId);
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("endSession call processed with no warnings or errors");
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("endSession call processed with a warning");
```

```
    }
    else
    {
        System.out.println("endSession call processed with an error");
    }
    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("endSession",
   response.getAdvisoryMessages());
```

executeBatch

The executeBatch method enables you to execute several methods with a single request to the runtime server.

executeBatch(String *sessionID*, *CommandImpl*[] commands)

- **sessionID**-A string identifying the session ID. This session ID is used for all commands run by this method call.
- **commandImpl[]**-An array of CommandImpl objects, one for each command you want to perform.

The result of calling this method is equivalent to explicitly calling each method in the Command array. This method minimizes the number of actual requests to the runtime server. The runtime server runs each method serially; for each call, any error or warnings are captured in the Response object that corresponds to that method call. If an error is encountered, the executeBatch continues with the rest of the calls in the batch. If the running of any method results in an error, the top level status for the BatchResponse object reflects that error. If no error occurred, the top level status reflects any warnings that may have occurred. If no warning occurred, then the top level status reflects a successful run of the batch.

Return value

The runtime server responds to the executeBatch with a BatchResponse object.

Example

The following example shows how to call all the getOffer and postEvent methods with a single executeBatch call, and a suggestion for how to handle the response.

```
/** Define all variables for all members of the executeBatch*/
String sessionId="MySessionID-123";
String interactionPoint = "Overview Page Banner 1";
int numberRequested=1;
String eventName = "logOffer";

/** build the getOffers command */
Command getOffersCommand = new CommandImpl();
getOffersCommand.setMethodIdentifier(Command.COMMAND_GETOFFERS);
getOffersCommand.setInteractionPoint(interactionPoint);
getOffersCommand.setNumberRequested(numberRequested);

/** build the postEvent command  */
Command postEventCommand = new CommandImpl();
postEventCommand.setMethodIdentifier(Command.COMMAND_POSTEVENT);
postEventCommand.setEventParameters(postEventParameters);
postEventCommand.setEvent(eventName);

/** Build command array */
Command[] commands =
{
  getOffersCommand,
  postEventCommand,
```

```
};

/** Make the call */
BatchResponse batchResponse = api.executeBatch(sessionId, commands);

/** Process the response appropriately */
// Top level status code is a short cut to determine if there
// are any non-successes in the array of Response objects
if(batchResponse.getBatchStatusCode() == Response.STATUS_SUCCESS)
{
  System.out.println("ExecuteBatch ran perfectly!");
}
else if(batchResponse.getBatchStatusCode() == Response.STATUS_WARNING)
{
  System.out.println("ExecuteBatch call processed with at least one warning");
}
else
{
  System.out.println("ExecuteBatch call processed with at least one error");
}

// Iterate through the array, and print out the message for any non-successes
for(Response response : batchResponse.getResponses())
{
  if(response.getStatusCode()!=Response.STATUS_SUCCESS)
  {
  printDetailMessageOfWarningOrError("executeBatchCommand",
 response.getAdvisoryMessages());
  }
}
```

Writing executeBatch() XML requests for the Interact SOAP API

Use these steps to write `executeBatch()` XML requests for the Interact SOAP API.

The request XML for a single operation SOAP API calls (`startSession`, `getOffers`, `setAudience`, `endSession`, and so on) must not be directly copied or pasted into a multiple operation `executeBatch()` call. The subcommands in the `executeBatch()` calls have slightly different WSDL and XML request structures than those of the single operation API calls. The structural differences cause failure responses from the server if the XML elements are copied and pasted from single operation API requests into multiple operation `executeBatch` requests.

Sample failure responses:

```
** XML Response Element: <ns0:faultstring>org.apache.axis2.databinding.ADBException:
Unexpected subelement audienceID</ns0:faultstring>
** Interact Server Exception: java.lang.Exception: org.apache.axis2.databinding.
ADBException: Unexpected subelement audienceID at
*** ... com.unicacorp.interact.api.soap.service.v1.xsd.CommandImpl$Factory.parse
(CommandImpl.java:1917) at
```

Use these steps to write an `executeBatch()` XML request. You can refer to single operation API call requests for parameter values during these steps, but do not copy and paste XML elements.

1. Use a WSDL processing tool (for example, SoapUI) to create a well-formed `executeBatch()` XML request from the Interact WSDL file.
2. Add subcommands to the request after the WSDL definition for `executeBatch()` child elements.
3. Complete the subcommand arguments after the WSDL definition for `executeBatch()` child elements.

getInstance

The `getInstance` method creates an instance of the Interact API that communicates with the specified runtime server.

```
getInstance(String URL)
```

Important: Every application you write using the Interact API must call `getInstance` to instantiate an InteractAPI object which is mapped to a runtime server specified by the URL parameter.

For server groups, if you are using a load balancer, use the hostname and port you configure with the load balancer. If you do not have a load balancer, you will have to include logic to rotate between the available runtime servers.

This method is applicable for the Java serialization over HTTP adapter only. There is no corresponding method defined in the SOAP WSDL. Each SOAP client implementation has its own way of establishing the endpoint URL.

- **URL** - A string identifying the URL for the runtime instance. For example, `http://localhost:7001/Interact/servlet/InteractJSService`.

Return value

The runtime server returns the InteractAPI.

Example

The following example shows how to instantiate an InteractAPI object that points to a runtime server instance running on the same machine as your touchpoint.

```
InteractAPI api=InteractAPI.getInstance("http://localhost:7001/interact/servlet/InteractJSService");
```

getOffers

The `getOffers` method enables you to request offers from the runtime server.

```
getOffers(String sessionID, String interactionPoint, int numberOfOffers)
```

- **sessionID**-a string identifying the current session.
- **interactionPoint**-a string identifying the name of the interaction point this method references.

  Note: This name must match the name of the interaction point defined in interactive channel exactly.
- **numberOfOffers**-an integer identifying the number of offers requested.

The `getOffers` method waits the number of milliseconds defined in the `segmentationMaxWaitTimeInMS` property for all re-segmentation to complete before running. Therefore, if you call a `postEvent` method which triggers a re-segmentation or a `setAudience` method immediately before a `getOffers` call, there may be a delay.

Return value

The runtime server responds to `getOffers` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList

- SessionID
- StatusCode

Example

This example shows requesting a single offer for the Overview Page Banner 1 interaction point and a way to handle the response.

sessionId is the same string to identify the runtime session used by the startSession call which started this session.

```
String interactionPoint = "Overview Page Banner 1";
int numberRequested=1;

/** Make the call */
response = api.getOffers(sessionId, interactionPoint, numberRequested);

/** Process the response appropriately */
 // check if response is successful or not
 if(response.getStatusCode() == Response.STATUS_SUCCESS)
 {
  System.out.println("getOffers call processed with no warnings or errors");

        /** Check to see if there are any offers */
        OfferList offerList=response.getOfferList();

        if(offerList.getRecommendedOffers() != null)
        {
            for(Offer offer : offerList.getRecommendedOffers())
            {
                // print offer
                System.out.println("Offer Name:"+offer.getOfferName());
            }
        }
        else // count on the default Offer String
        System.out.println("Default offer:"+offerList.getDefaultString());
   }
   else if(response.getStatusCode() == Response.STATUS_WARNING)
   {
       System.out.println("getOffers call processed with a warning");
   }
   else
   {
       System.out.println("getOffers call processed with an error");
   }
   // For any non-successes, there should be advisory messages explaining why
   if(response.getStatusCode() != Response.STATUS_SUCCESS)
       printDetailMessageOfWarningOrError("getOffers",
  response.getAdvisoryMessages());
```

getOffersForMultipleInteractionPoints

The getOffersForMultipleInteractionPoints method enables you to request offers from the runtime server for multiple IPs with deduplication.

getOffersForMultipleInteractionPoints(String *sessionID*, String *requestStr*)

- **sessionID** - a string identifying the current session.
- **requestStr** - a string providing an array of GetOfferRequest objects.

  Each GetOfferRequest object specifies:
  - **ipName** - The interaction point (IP) name for which the object is requesting offers
  - **numberRequested** - The number of unique offers it needs for the specified IP

– **offerAttributes** - Requirements on the attributes of the delivered offers using an instance of `OfferAttributeRequirements`
– **duplicationPolicy** - Duplication policy ID for the offers that will be delivered

  Duplication policies determine whether duplicated offers will be returned across different interaction points in a single method call. (*Within* an individual interaction point, duplicated offers are never returned.) Currently, two duplication policies are supported.

  - NO_DUPLICATION (ID value = 1). None of the offers that have been included in the preceding `GetOfferRequest` instances will be included in this `GetOfferRequest` instance (that is, Interact will apply de-duplication).
  - ALLOW_DUPLICATION (ID value = 2). Any of the offers satisfying the requirements specified in this `GetOfferRequest` instance will be included. The offers that have been included in the preceding `GetOfferRequest` instances will not be reconciled.

  The order of requests in the array parameter is also the priority order when offers are being delivered.

  For example, suppose the IPs in the request are IP1, then IP2, that no duplicated offers are allowed (a duplication policy ID = 1), and each is requesting two offers. If Interact finds offers A, B, and C for IP1 and offers A and D for IP2, the response will contain offers A and B for IP1, and only offer D for IP2.

  Also note that when the duplication policy ID is 1, the offers that have been delivered via an IP with higher priority will not be delivered via this IP.

The `getOffersForMultipleInteractionPoints` method waits the number of milliseconds defined in the `segmentationMaxWaitTimeInMS` property for all re-segmentation to complete before running. Therefore, if you call a `postEvent` method which triggers a re-segmentation or a `setAudience` method immediately before a `getOffers` call, there may be a delay.

Return value

The runtime server responds to `getOffersForMultipleInteractionPoints` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- array of OfferList
- SessionID
- StatusCode

Example

```
InteractAPI api = InteractAPI.getInstance("url");
 String sessionId = "123";
 String requestForIP1 = "{IP1,5,1,(5,attr1=1|numeric;attr2=value2|string,
     (3,attr3=value3|string)(3,attr4=4|numeric))}";
 String requestForIP2 = "{IP2,3,2,(3,attr5=value5|string)}";
 String requestForIP3 = "{IP3,2,1}";
 String requestStr = requestForIP1 + requestForIP2 + requestForIP3;
 Response response = api.getOffersForMultipleInteractionPoints(sessionId,
          requestStr);

 if (response.getStatusCode() == Response.STATUS_SUCCESS) {
  // Check to see if there are any offers
```

```
  OfferList[] allOfferLists = response.getAllOfferLists();
  if (allOfferLists != null) {
   for (OfferList ol : allOfferLists) {
    System.out.println
("The following offers are delivered for interaction
               point " + ol.getInteractionPointName() + ":");
    for (Offer o : ol.getRecommendedOffers()) {
     System.out.println(o.getOfferName());
    }
   }
  }
 }
 else {
  System.out.println("getOffersForMultipleInteractionPoints() method calls
          returns an error with code: " + response.getStatusCode());
 }
```

Note that the syntax of the `requestStr` is the following:

```
requests_for_IP[<requests_for_IP]
```

where

```
<requests_for_IP> = {ip_name,number_requested_for_this_ip,
     dupe_policy[,child_requirements]]}
attribute_requirements = (number_requested_for_these_attribute_requirements
     [,attribute_requirement[;individual_attribute_requirement])
     [,(attribute_requirements))
individual_attribute_requirement = attribute_name=attribute_value | attribute_type
```

In the example shown above, `requestForIP1 ({IP1,5,1,(5,attr1=1|numeric; attr2=value2|string, (3,attr3=value3|string)(3,attr4=4|numeric))})` means, for the interaction point named IP1, deliver at most 5 distinct offers that can not also be returned for any other interaction points during this same method call. All of those 5 offers must have a numeric attribute named `attr1` which must have the value 1, and must have a string attribute named `attr2` which must have the value *value2*. Out of those 5 offers, a maximum of 3 must have a string attribute named `attr3` which must have the value *value3*, and a maximum of 3 must have a numeric attribute named `attr4` which must have the value *4*.

The allowed attribute types are numeric, string, and datetime, and the format of a datetime attribute value must be `MM/dd/yyyy HH:mm:ss`. To retrieve the returned offers, use the method `Response.getAllOfferLists()`. To help understand the syntax, the example in `setGetOfferRequests` builds the same instance of `GetOfferRequests`, while using Java objects, which is preferred.

getProfile

The `getProfile` method enables you to retrieve the profile and temporal information about the visitor visiting the touchpoint.

`getProfile(String sessionID)`

- **sessionID**-a string identifying the session ID.

Return value

The runtime server responds to `getProfile` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion

- ProfileRecord
- SessionID
- StatusCode

Example

The following is an example of using `getProfile` and a way to handle the response.

`sessionId` is the same string to identify the session used by the `startSession` call which started this session.

```
response = api.getProfile(sessionId);
/**  Process the response appropriately */
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("getProfile call processed with no warnings or errors");
        // Print the profile - it's just an array of NameValuePair objects
        for(NameValuePair nvp : response.getProfileRecord())
        {
            System.out.println("Name:"+nvp.getName());
            if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_DATETIME))
            {
                System.out.println("Value:"+nvp.getValueAsDate());
            }
            else if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_NUMERIC))
            {
                System.out.println("Value:"+nvp.getValueAsNumeric());
            }
            else
            {
                System.out.println("Value:"+nvp.getValueAsString());
            }
        }
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("getProfile call processed with a warning");
    }
    else
    {
        System.out.println("getProfile call processed with an error");
    }
    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("getProfile",
   response.getAdvisoryMessages());
```

getVersion

The `getVersion` method returns the version of the current implementation of the Interact runtime server.

```
getVersion()
```

Best practice is to use this method when you initialize the touchpoint with the Interact API.

Return value

The runtime server responds to the `getVersion` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- StatusCode

Example

This example shows a simple way to call `getVersion` and process the results.

```
response = api.getVersion();
/** Process the response appropriately */
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("getVersion call processed with no warnings or errors");
        System.out.println("API Version:" + response.getApiVersion(););
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("getVersion call processed with a warning");
    }
    else
    {
        System.out.println("getVersion call processed with an error");
    }

    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("getVersion",
   response.getAdvisoryMessages());
```

postEvent

The `postEvent` method enables you to execute any event defined in the interactive channel.

`postEvent(String` *`sessionID`*`, String` *`eventName`*`, NameValuePairImpl[]` *`eventParameters`*`)`

- **sessionID**: a string identifying the session ID.
- **eventName**: a string identifying the name of the event.

  Note: The name of the event must match the name of the event as defined in the interactive channel. This name is case-insensitive.
- **eventParameters**. `NameValuePairImpl` objects identifying any parameters that need to be passed with the event. These values are stored in the session data.

  If this event triggers re-segmentation, you must ensure that all data required by the interactive flowcharts is available in the session data. If any of these values have not been populated by prior actions (for example, from `startSession` or `setAudience`, or loading the profile table) you must include an eventParameter for every missing value. For example, if you have configured all profile tables to load into memory, you must include a NameValuePair for temporal data required for the interactive flowcharts.

  If you are using more than one audience level, you most likely have different sets of eventParameters for each audience level. You should include some logic to ensure you are selecting the correct set of parameters for the audience level.

  Important: If this event logs to response history, you must pass the treatment code for the offer. You must define the name for the NameValuePair as `"UACIOfferTrackingCode"`.

You can only pass one treatment code per event. If you do not pass the treatment code for an offer contact, Interact logs an offer contact for every offer in the last recommended list of offers. If you do not pass the treatment code for a response, Interact returns an error.

- There are several other reserved parameters used with `postEvent` and other methods and are discussed later in this section.

Any request for re-segmentation or writing to contact or response history does not wait for a response.

Re-segmentation does not clear prior segmentation results for the current audience level. You can use the `UACIExecuteFlowchartByName` parameter to define specific flowcharts to run. The `getOffers` method waits for re-segmentation to finish before running. Therefore, if you call a `postEvent` method, which triggers a re-segmentation immediately before a `getOffers` call, there might be a delay.

Return value

The runtime server responds to `postEvent` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- SessionID
- StatusCode

Example

The following `postEvent` example shows sending new parameters for an event which triggers re-segmentation, and a way to handle the response.

`sessionId` is the same string to identify the session used by the `startSession` call which started this session.

```
String eventName = "SearchExecution";

NameValuePair parmB1 = new NameValuePairImpl();
parmB1.setName("SearchString");
parmB1.setValueAsString("mortgage");
parmB1.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parmB2 = new NameValuePairImpl();
parmB2.setName("TimeStamp");
parmB2.setValueAsDate(new Date());
parmB2.setValueDataType(NameValuePair.DATA_TYPE_DATETIME);

NameValuePair parmB3 = new NameValuePairImpl();
parmB3.setName("Browser");
parmB3.setValueAsString("IE6");
parmB3.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parmB4 = new NameValuePairImpl();
parmB4.setName("FlashEnabled");
parmB4.setValueAsNumeric(1.0);
parmB4.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

NameValuePair parmB5 = new NameValuePairImpl();
parmB5.setName("TxAcctValueChange");
parmB5.setValueAsNumeric(0.0);
parmB5.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
```

```
NameValuePair parmB6 = new NameValuePairImpl();
parmB6.setName("PageTopic");
parmB6.setValueAsString("");
parmB6.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair[] postEventParameters = { parmB1,
 parmB2,
 parmB3,
 parmB4,
 parmB5,
 parmB6
};

/** Make the call */
response = api.postEvent(sessionId, eventName, postEventParameters);

/** Process the response appropriately */
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("postEvent call processed with no warnings or errors");
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("postEvent call processed with a warning");
    }
    else
    {
        System.out.println("postEvent call processed with an error");
    }

    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("postEvent",
   response.getAdvisoryMessages());
```

setAudience

The `setAudience` method enables you to set the audience ID and level for a visitor.

```
setAudience(String sessionID, NameValuePairImpl[] audienceID,
     String audienceLevel, NameValuePairImpl[] parameters)
```

- **sessionID** - a string identifying the session ID.
- **audienceID** - an array of `NameValuePairImpl` objects that defines the audience ID.
- **audienceLevel** - a string that defines the audience level.
- **parameters** - `NameValuePairImpl` objects identifying any parameters that need to be passed with `setAudience`. These values are stored in the session data and can be used for segmentation.

  You must have a value for every column in your profile. This is a superset of all columns in all the tables defined for the interactive channel and any real-time data. If you have already populated all the session data with `startSession` or `postEvent`, you do not need to send new parameters.

The `setAudience` method triggers a re-segmentation. The `getOffers` method waits for re-segmentation to finish before running. Therefore, if you call a `setAudience` method immediately before a `getOffers` call, there may be a delay.

The `setAudience` method also loads the profile data for the audience ID. You can use the `setAudience` method to force a reload of the same profile data loaded by the `startSession` method.

Return value

The runtime server responds to `setAudience` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- SessionID
- StatusCode

Example

For this example, the audience level stays the same, but the ID changes, as if an anonymous user logs in and becomes known.

`sessionId` and `audienceLevel` are the same strings to identify the session and audience level used by the `startSession` call which started this session.

```
NameValuePair custId2 = new NameValuePairImpl();
custId2.setName("CustomerId");
custId2.setValueAsNumeric(123.0);
custId2.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

NameValuePair[] newAudienceId = { custId2 };

/** Parameters can be passed in as well.  For this example, there are no parameters,
* therefore pass in null */
NameValuePair[] noParameters=null;

/** Make the call  */
response = api.setAudience(sessionId, newAudienceId, audienceLevel, noParameters);

 /** Process the response appropriately */
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("setAudience call processed with no warnings or errors");
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("setAudience call processed with a warning");
    }
    else
    {
        System.out.println("setAudience call processed with an error");
    }

    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("setAudience",
   response.getAdvisoryMessages());
```

setDebug

The `setDebug` method enables you to set the logging verbosity level for all code paths for the session.

`setDebug(String` *`sessionID`*`, boolean` *`debug`*`)`

- **sessionID**-a string which identifies the session ID.
- **debug**-a boolean which enables or disables debug information. Valid values are `true` or `false`. If true, Interact logs debug information to the runtime server log.

Return value

The runtime server responds to `setDebug` with a Response object with the following attributes populated:

- AdvisoryMessages

- ApiVersion
- SessionID
- StatusCode

Example

The following example shows changing the debug level of the session.

sessionId is the same string to identify the session used by the startSession call which started this session.

```
boolean newDebugFlag=false;
/** make the call */
response = api.setDebug(sessionId, newDebugFlag);

/** Process the response appropriately */
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("setDebug call processed with no warnings or errors");
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("setDebug call processed with a warning");
    }
    else
    {
        System.out.println("setDebug call processed with an error");
    }

    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("setDebug",
   response.getAdvisoryMessages());
```

startSession

The startSession method creates and defines a runtime session.

```
startSession(String sessionID,
   boolean relyOnExistingSession,
   boolean debug,
   String interactiveChannel,
   NameValuePairImpl[] audienceID,
   String audienceLevel,
   NameValuePairImpl[] parameters)
```

startSession can trigger up to five actions:

- create a runtime session.
- load visitor profile data for the current audience level into the runtime session, including any dimension tables marked for loading in the table mapping defined for the interactive channel.
- trigger segmentation, running all interactive flowcharts for the current audience level.
- load offer suppression data into the session, if the enableOfferSuppressionLookup property is set to true.
- load score override data into the session, if the enableScoreOverrideLookup property is set to true.

The startSession method requires the following parameters:

- **sessionID**-a string which identifies the session ID. You must define the session ID. For example, you could use a combination of customer ID and timestamp.

  To define what makes a runtime session, a session id has to be specified. This value is managed by the client. All method calls for the same session id has to be synchronized by the client. The behavior for concurrent API calls with the same session id is undefined.
- **relyOnExistingSession** - a boolean which defines whether this session uses a new or an existing session. Valid values are `true` or `false`. If `true`, you must supply an existing session ID with the `startSession` method. If `false`, you must supply a new session ID.

  If you set `relyOnExistingSession` to `true` and a session exists, the runtime environment uses the existing session data and does not reload any data or trigger segmentation. If the session does not exist, the runtime environment creates a new session, including loading data and triggering segmentation. Setting `relyOnExistingSession` to true and using it with all `startSession` calls is useful if your touchpoint has a longer session length than the runtime session. For example, a web site session is alive for 2 hours, but the runtime session is only alive for 20 minutes.

  If you call `startSession` twice with the same session ID, all session data from the first `startSession` call is lost if `relyOnExistingSession` is false.
- **debug** - a boolean which enables or disables debug information. Valid values are `true` or `false`. If true, Interact logs debug information to the runtime server logs. The debug flag is set for each session individually. Therefore, you can trace debug data for an individual session.
- **interactiveChannel**-a string defining the name of the interactive channel this session refers to. This name must match the name of the interactive channel defined in Campaign exactly.
- **audienceID** - an array of NameValuePairImpl objects where the names must match the physical column names of any table containing the audience ID.
- **audienceLevel** - a string defining the audience level.
- **parameters** - NameValuePairImpl objects identifying any parameters that need to be passed with `startSession`. These values are stored in the session data and can be used for segmentation.

  If you have several interactive flowcharts for the same audience level, you must include a superset of all columns in all the tables. If you configure the runtime to load the profile table, and the profile table contains all the columns you require, you do not need to pass any parameters, unless you want to overwrite the data in the profile table. If your profile table contains a subset of the required columns, you must include the missing columns as parameters.

If the `audienceID` or `audienceLevel` are invalid and `relyOnExistingSession` is false, the `startSession` call fails. If the `interactiveChannel` is invalid, `startSession` fails, whether `relyOnExistingSession` is true or false.

If `relyOnExistingSession` is true, and you make a second `startSession` call using the same `sessionID`, but the first session has expired, Interact creates a new session.

If `relyOnExistingSession` is true, and you make a second `startSession` call using the same `sessionID` but a different `audienceID` or `audienceLevel`, the runtime server changes the audience for the existing session.

If relyOnExistingSession is true, and you make a second startSession call using the same sessionID but a different interactiveChannel, the runtime server creates a new session.

Return value

The runtime server responds to startSession with a Response object with the following attributes populated:

- AdvisoryMessages (if StatusCode does not equal 0)
- ApiVersion
- SessionID
- StatusCode

Example

The following example shows one way to call startSession.

```
String sessionId="MySessionID-123";
String audienceLevel="Customer";
NameValuePair custId = new NameValuePairImpl();
custId.setName("CustomerId");
custId.setValueAsNumeric(1.0);
custId.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
NameValuePair[] initialAudienceId = { custId };
boolean relyOnExistingSession=false;
boolean initialDebugFlag=true;
String interactiveChannel="Accounts Website";
NameValuePair parm1 = new NameValuePairImpl();
parm1.setName("SearchString");
parm1.setValueAsString("");
parm1.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parm2 = new NameValuePairImpl();
parm2.setName("TimeStamp");
parm2.setValueAsDate(new Date());
parm2.setValueDataType(NameValuePair.DATA_TYPE_DATETIME);

NameValuePair parm3 = new NameValuePairImpl();
parm3.setName("Browser");
parm3.setValueAsString("IE6");
parm3.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parm4 = new NameValuePairImpl();
parm4.setName("FlashEnabled");
parm4.setValueAsNumeric(1.0);
parm4.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

NameValuePair parm5 = new NameValuePairImpl();
parm5.setName("TxAcctValueChange");
parm5.setValueAsNumeric(0.0);
parm5.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

NameValuePair parm6 = new NameValuePairImpl();
parm6.setName("PageTopic");
parm6.setValueAsString("");
parm6.setValueDataType(NameValuePair.DATA_TYPE_STRING);

/** Specifying the parameters (optional) */
NameValuePair[] initialParameters = { parm1,
 parm2,
 parm3,
 parm4,
 parm5,
```

```
 parm6
 };

/** Make the call */
response = api.startSession(sessionId, relyOnExistingSession, initialDebugFlag,
 interactiveChannel, initialAudienceId, audienceLevel, initialParameters);

/** Process the response appropriately */
processStartSessionResponse(response);
```

processStartSessionResponse is a method which handles the response object returned by startSession.

```
public static void processStartSessionResponse(Response response)
{
  // check if response is successful or not
  if(response.getStatusCode() == Response.STATUS_SUCCESS)
 {
  System.out.println("startSession call processed with no warnings or errors");
 }
  else if(response.getStatusCode() == Response.STATUS_WARNING)
 {
  System.out.println("startSession call processed with a warning");
 }
  else
 {
  System.out.println("startSession call processed with an error");
 }

 // For any non-successes, there should be advisory messages explaining why
 if(response.getStatusCode() != Response.STATUS_SUCCESS)
  printDetailMessageOfWarningOrError("StartSession",
   response.getAdvisoryMessages());
}
```

Offer deduplication across offer attributes

Using the Interact application programming interface (API), two API calls deliver offers: getOffers and getOffersForMultipleInteractionPoints. getOffersForMultipleInteractionPoints can prevent the return of duplicate offers at the *OfferID* level, but cannot deduplicate offers across offer category. So, for example, for Interact to return only one offer from each offer category, a workaround was previously required. With the introduction of two parameters to the startSession API call, offer deduplication across offer attributes, such as category, is now possible.

This list summarizes the parameters that were added to the startSession API call. For more information about these parameters or any aspect of the Interact API, see the *IBM Interact Administrator's Guide*, or the Javadoc files included with your Interact installation in <Interact_Home>/docs/apiJavaDoc.

- UACIOfferDedupeAttribute. To create a startSession API call with offer deduplication, so that the subsequent getOffer calls return only one offer from each category, you must include the UACIOfferDedupeAttribute parameter as part of the API call. You can specify a parameter in the name,value,type format, as shown here:

  UACIOfferDedupeAttribute,<attributeName>,string

  In this example, you would replace *<attributeName>* with the name of the offer attribute you want to use as the criterion for deduplication, such as Category.

  Note: Interact examines the offers that have the same attribute value you specify (such as Category) and deduplicate to remove all but the offer that has the

highest score. If the offers that have the duplicate attribute also have identical scores, Interact returns a random selection from among the matching offers.

- `UACINoAttributeDedupeIfFewerOf`. When you include the `UACIOfferDedupeAttribute` in the `startSession` call, you can also set this `UACINoAttributeDedupeIfFewerOf` parameter to specify the behavior in cases where the offer list after deduplication no longer contains enough offers to satisfy the original request.

  For example, if you set `UACIOfferDedupeAttribute` to use the offer category to deduplicate offers, and your subsequent getOffers call requests that eight offers be returned, deduplication might result in fewer than eight eligible offers. In that case, setting `UACINoAttributeDedupeIfFewerOf` parameter to true would result in adding some of the duplicated to the eligible list to satisfy the requested number of offers. In this example, if you set the parameter to false, the number of offers that are returned would be fewer than the requested number.

  `UACINoAttributeDedupeIfFewerOf` is set to true by default.

For example, suppose you specified as a `startSession` parameter that the deduplication criterion is offer Category, as shown here:

```
UACIOfferDedupeAttribute, Category,
string;UACINoAttributeDedupeIfFewerOffer, 0, string
```

These parameters together cause Interact to deduplicate offers based on the offer attribute "Category," and to return only the deduplicated offers even if the resulting number of offers is fewer than requested (`UACINoAttributeDedupeIfFewerOffer` is false).

When you issue a `getOffers` API call, the original set of eligible offers might include these offers:

- Category=Electronics: Offer A1 with a score of 100 and Offer A2 with a score of 50.
- Category=Smartphones: Offer B1 with a score of 100, Offer B2 with a score of 80, and offer B3 with a score of 50.
- Category=MP3Players: Offer C1 with a score of 100, Offer C2 with a score of 50.

In this case, there were two duplicate offers that match the first category, three duplicate offers that match the second category, and two duplicate offers that match the third category. The offers that are returned would be the highest scoring offers from each category, which are Offer A1, Offer B1, and Offer C1.

If the `getOffers` API call requested six offers, this example set `UACINoAttributeDedupeIfFewerOffer` to false, so only three offers would be returned.

If the `getOffers` API call requested six offers, and this example omitted the `UACINoAttributeDedupeIfFewerOffer` parameter, or specifically set it to true, some of the duplicate offers would be included in the result to satisfy the requested number.

Reserved parameters

There are several reserved parameters used with the Interact API. Some are required for the runtime server, and others you can use for additional features.

postEvent features

Feature	Parameter	Description
Log to custom table	UACICustomLoggerTableName	The name of a table in the runtime tables data source. If you provide this parameter with a valid table name, the runtime environment writes all session data to the selected table. All column names in the table that match session data NameValuePair are populated. The runtime environment populates any column that does not match a session name-value pair with a null. You can manage the process which writes to the database with the customLogger configuration properties.
Multiple response types	UACILogToLearning	An integer with the value 1 or 0. 1 indicates the runtime environment should log the event as an accept to the learning system or enable offer suppression within a session. 0 indicates the runtime environment should not log the event to the learning system or enable offer suppression within a session. This parameter enables you to create several postEvent methods logging different response types without influencing learning. You do not need to define this parameter for events set to log a contact, accept, or reject. You must use this parameter in conjunction with UACIResponseTypeCode. If you do not define UACILOGTOLEARNING, the runtime environment assumes the default value of 0 (unless the event triggers a log contact, accept, or reject).
	UACIResponseTypeCode	A value representing a response type code. The value must be a valid entry in the UA_UsrResponseType table
Response tracking	UACIOfferTrackingCode	The treatment code for the offer. You must define this parameter if the event logs to contact or response history. You can only pass one treatment code per event. If you do not pass the treatment code for an offer contact, the runtime environment logs an offer contact for every offer in the last recommended list of offers. If you do not pass the treatment code for a response, the runtime environment returns an error. If you configure the cross-session response tracking, you can use the UACIOfferTrackingcodeType parameter to define what type of tracking code you use other than treatment code.
Cross-session response tracking	UACIOfferTrackingCodeType	A number which defines the tracking code type. 1 is the default treatment code, and 2 is the offer code. All codes must be valid entries in the UACI_TrackingType table. You can add other, custom codes to this table.

Feature	Parameter	Description
Specific flowchart execution	UACIExecuteFlowchartByName	If you define this parameter for any method which triggers segmentation (startSession, setAudience, or a postEvent that triggers re-segmentation), instead of running all flowcharts for the current audience level, Interact runs only the named flowcharts. You can provide a list of flowcharts separated by a pipe (|) character.

runtime environment reserved parameters

The following reserved parameters are used by the runtime environment. Do not use these names for your event parameters.

- UACIEventID
- UACIEventName
- UACIInteractiveChannelID
- UACIInteractiveChannelName
- UACIInteractionPointID
- UACIInteractionPointName
- UACISessionID

About the AdvisoryMessage class

The advisoryMessage class contains methods which define the advisory message object. The advisory message object is contained in the Response object. Every method in the InteractAPI returns a Response object. (Except for the executeBatch method, which returns a batchResponse object.)

If there is an error or a warning, the Interact server populates the advisory message object. The advisory message object contains the following attributes:

- **DetailMessage**-a verbose description of the advisory message. This attribute may not be available for all advisory messages. If available, the DetailMessage may not be localized.
- **Message**-a short description of the advisory message.
- **MessageCode**-a code number for the advisory message.
- **StatusLevel**-a code number for the severity of the advisory message.

You retrieve the advisoryMessage objects by using the getAdvisoryMessages method.

getDetailMessage

The getDetailMessage method returns the detailed, verbose description of an Advisory Message object. Not all messages have a detailed message.

```
getDetailMessage()
```

Return value

The Advisory Message object returns a string.

Example

```
// For any non-successes, there should be advisory messages explaining why
if(response.getStatusCode() != Response.STATUS_SUCCESS)
{
 for(AdvisoryMessage msg : response.getAdvisoryMessages())
 {
  System.out.println(msg.getMessage());
  // Some advisory messages may have additional detail:
  System.out.println(msg.getDetailMessage());
 }
}
```

getMessage

The getMessage method returns the brief description of an Advisory Message object.

```
getMessage()
```

Return value

The Advisory Message object returns a string.

Example

The following method prints out the message and detailed message of an AdvisoryMessage object.

```
// For any non-successes, there should be advisory messages explaining why
if(response.getStatusCode() != Response.STATUS_SUCCESS)
{
 for(AdvisoryMessage msg : response.getAdvisoryMessages())
 {
  System.out.println(msg.getMessage());
  // Some advisory messages may have additional detail:
  System.out.println(msg.getDetailMessage());
 }
}
```

getMessageCode

The getMessageCode method returns the internal error code associated with an Advisory Message object if the status level is 2 (STATUS_LEVEL_ERROR).

```
getMessageCode()
```

Return value

The AdvisoryMessage object returns an integer.

Example

The following method prints out the message code of an AdvisoryMessage object.

```
public static void printMessageCodeOfWarningOrError(String command,AdvisoryMessage[] messages)
    {
        System.out.println("Calling "+command);
        for(AdvisoryMessage msg : messages)
        {
            System.out.println(msg.getMessageCode());
```

getStatusLevel

The getStatusLevel method returns the status level of an Advisory Message object.

```
getStatusLevel()
```

Return value

The Advisory Message object returns an integer.

- 0 - STATUS_LEVEL_SUCCESS-The method called completed with no errors.
- 1 - STATUS_LEVEL_WARNING-The method called completed with at least one warning (but no errors).
- 2 - STATUS_LEVEL_ERROR-The method called did not complete successfully and has at least one error.

Example

The following method prints out the status level of an AdvisoryMessage object.

```
public static void printMessageCodeOfWarningOrError(String command,AdvisoryMessage[] messages)
    {
        System.out.println("Calling "+command);
        for(AdvisoryMessage msg : messages)
        {
            System.out.println(msg.getStatusLevel());
```

About the AdvisoryMessageCode class

The `advisoryMessageCode` class contains methods which define the advisory message codes. You retrieve the advisory message codes with the `getMessageCode` method.

Advisory message codes

You retrieve the advisory message codes with the `getMessageCode` method.

This table lists and describes the advisory message codes.

Code	Message text	Description
1	INVALID_SESSION_ID	The session ID does not reference a valid session.
2	ERROR_TRYING_TO_ABORT_ SEGMENTATION	An error occurred while trying to abort segmentation during an `endSession` call.
3	INVALID_INTERACTIVE_CHANNEL	The argument that was passed in for interactive channel does not reference a valid interactive channel.
4	INVALID_EVENT_NAME	The argument that was passed in for the event does not reference a valid event for the current interactive channel.
5	INVALID_INTERACTION_POINT	The argument that was passed in for the interaction point does not reference a valid interaction point for the current interactive channel.
6	ERROR_WHILE_MAKING_INITIAL_ SEGMENTATION_REQUEST	An error was encountered when submitting a segmentation request.
7	SEGMENTATION_RUN_FAILED	The segmentation ran partly, but resulted in an error.
8	PROFILE_LOAD_FAILED	The attempt to load the profile or dimension tables failed.
9	OFFER_SUPPRESSION_LOAD_FAILED	The attempt to load the offer suppression table failed.

Code	Message text	Description
10	COMMAND_METHOD_ UNRECOGNIZED	A command method that was specified for a command within an `executeBatch` call is not valid.
11	ERROR_TRYING_TO_POST_EVENT _PARAMETERS	An error occurred while posting the event parameters.
12	LOG_SYSTEM_EVENT_EXCEPTION	An exception occurred when trying to submit system event (End Session, Get Offer, Get Profile, Set Audience, Set Debug, or Start Session) for logging.
13	LOG_USER_EVENT_EXCEPTION	An exception occurred when trying to submit user event for logging.
14	ERROR_TRYING_TO_LOOK_UP_EVENT	An error occurred when trying to look up the event name.
15	ERROR_TRYING_TO_LOOK_UP _INTERACTIVE_CHANNEL	An error occurred when trying to look up the interactive channel name.
16	ERROR_TRYING_TO_LOOK_UP _INTERACTION_POINT	An error occurred when trying to look up the interaction point name.
17	RUNTIME_EXCEPTION_ ENCOUNTERED	An unexpected runtime exception was encountered.
18	ERROR_TRYING_TO_EXECUTE_ ASSOCIATED_ACTION	An error occurred while trying to run associated action (Trigger Resegmentation, Log Offer Contact, Log Offer Acceptance, or Log Offer Rejection).
19	ERROR_TRYING_RUN_FLOWCHART	An error occurred while trying to run flowchart.
20	FLOWCHART_FAILED	A flowchart run failed.
21	FLOWCHART_ABORTED	A flowchart run was aborted.
22	FLOWCHART_NEVER_RUN	A specified flowchart was never run.
23	FLOWCHART_STILL_RUNNING	A flowchart is still running.
24	ERROR_WHILE_READING_ PARAMETERS	An error occurred while reading parameters.
25	ERROR_WHILE_LOADING_ RECOMMENDED_OFFERS	Error while loading recommended offers
26	ERROR_WHILE_LOGGING_DEFAULT _TEXT_STATISTICS	An error occurred while logging default text statistics (the number of times the default string for the interaction point displayed).
27	SCORE_OVERRIDE_LOAD_FAILED	The score override table failed to load.
28	NULL_AUDIENCE_ID	The audience identifier is empty.
29	UNRECOGNIZED_AUDIENCE_LEVEL	An unrecognized audience level was specified.
30	MISSING_AUDIENCE_FIELD	An audience field is missing.
31	INVALID_AUDIENCE_FIELD_TYPE	An invalid audience field type was specified.
32	UNSUPPORTED_AUDIENCE_FIELD _TYPE	Unsupported audience field type

Code	Message text	Description
33	TIMEOUT_REACHED_ON_GET_ OFFERS_CALL	The `getOffers` call reached timeout without returning offers.
34	INTERACT_INITIALIZATION_NOT_ COMPLETED_SUCCESSFULLY	The runtime server initialization did not complete successfully.
35	SESSION_ID_UNDEFINED	The session identifier is undefined.
36	INVALID_NUMBER_OF_OFFERS_ REQUESTED	An invalid number of offers was requested.
37	NO_SESSION_EXIST_BUT_WILL_ CREATE_NEW_ONE	No session existed, but one was created.
38	AUDIENCE_ID_NOT_FOUND_IN_ PROFILE_TABLE	The specified audience identifier is not in the profile table.
39	LOG_CUSTOM_LOGGER_EVENT_ EXCEPTION	An exception occurred when trying to submit custom logging data event.
40	SPECIFIED_FLOWCHART_FOR_ EXECUTION_DOES_NOT_EXIST	The specified flowchart cannot be run because it does not exist.
41	AUDIENCE_NOT_DEFINED_IN_ CONFIGURATION	The specified audience is not defined in the configuration.

About the BatchResponse class

The `BatchResponse` class contains methods which define the results of the `executeBatch` method.

The Batch Response object contains the following attributes:

- **BatchStatusCode**-The highest Status Code value for all the responses requested by the `executeBatch` method.
- **Responses**-An array of the Response objects requested by the `executeBatch` method.

getBatchStatusCode

The `getBatchStatusCode` method returns the highest status code from the array of commands executed by the `executeBatch` method.

```
getBatchStatusCode()
```

Return value

The `getBatchStatusCode` method returns an integer.

- 0 - STATUS_SUCCESS-The method called completed with no errors.
- 1 - STATUS_WARNING-The method called completed with at least one warning (but no errors).
- 2 - STATUS_ERROR-The method called did not complete successfully and has at least one error.

Example

The following code sample gives an example of how to retrieve the `BatchStatusCode`.

```
// Top level status code is a short cut to determine if there are any
// non-successes in the array of Response objects
if(batchResponse.getBatchStatusCode() == Response.STATUS_SUCCESS)
```

```
{
  System.out.println("ExecuteBatch ran perfectly!");
}
else if(batchResponse.getBatchStatusCode() == Response.STATUS_WARNING)
{
  System.out.println("ExecuteBatch call processed with at least one warning");
}
else
{
  System.out.println("ExecuteBatch call processed with at least one error");
}

// Iterate through the array, and print out the message for any non-successes
for(Response response : batchResponse.getResponses())
{
  if(response.getStatusCode()!=Response.STATUS_SUCCESS)
  {
      printDetailMessageOfWarningOrError("executeBatchCommand",
   response.getAdvisoryMessages());
  }
}
```

getResponses

The `getResponses` method returns the array of response objects that correspond to the array of commands executed by the `executeBatch` method.

```
getResponses()
```

Return value

The `getResponses` method returns an array of `Response` objects.

Example

The following example selects all the responses and prints out any advisory messages if the command was not successful.

```
for(Response response : batchResponse.getResponses())
{
    if(response.getStatusCode()!=Response.STATUS_SUCCESS)
    {
        printDetailMessageOfWarningOrError("executeBatchCommand",
   response.getAdvisoryMessages());
    }
}
```

About the Command interface

The `executeBatch` method requires you to pass in an array of objects that implements the Command interface. You should use the default implementation, `CommandImpl` to pass in the Command objects.

The following table lists the command, the method of the InteractAPI class the command represents, and the Command interface methods you must use for each command. You do not need to include a session ID because the `executeBatch` method already includes the session ID.

Command	Interact API Method	Command Interface Methods
COMMAND_ENDSESSION	endSession	None.

Command	Interact API Method	Command Interface Methods
COMMAND_GETOFFERS	getOffers	• setInteractionPoint • setNumberRequested
COMMAND_GETPROFILE	getProfile	None.
COMMAND_GETVERSION	getVersion	None.
COMMAND_POSTEVENT	postEvent	• setEvent • setEventParameters
COMMAND_SETAUDIENCE	setAudience	• setAudienceID • setAudienceLevel • setEventParameters
COMMAND_SETDEBUG	setDebug	setDebug
COMMAND_STARTSESSION	startSession	• setAudienceID • setAudienceLevel • setDebug • setEventParameters • setInteractiveChannel • setRelyOnExistingSession

setAudienceID

The `setAudienceID` method defines the AudienceID for the `setAudience` and `startSession` commands.

`setAudienceID(`*`audienceID`*`)`

- **audienceID**-an array of NameValuePair objects which define the AudienceID.

Return value

None.

Example

The following example is an excerpt from an `executeBatch` method calling `startSession` and `setAudience`.

```
NameValuePair custId = new NameValuePairImpl();
custId.setName("CustomerId");
custId.setValueAsNumeric(1.0);
custId.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
NameValuePair[] initialAudienceId = { custId };
. . .
Command startSessionCommand = new CommandImpl();
startSessionCommand.setAudienceID(initialAudienceId);
. . .
Command setAudienceCommand = new CommandImpl();
setAudienceCommand.setAudienceID(newAudienceId);
. . .
/** Build command array */
Command[] commands =
    {
        startSessionCommand,
        setAudienceCommand,
    };
/** Make the call */
```

```
    BatchResponse batchResponse = api.executeBatch(sessionId, commands);

/** Process the response appropriately */
    processExecuteBatchResponse(batchResponse);
```

setAudienceLevel

The `setAudienceLevel` method defines the Audience Level for the `setAudience` and `startSession` commands.

`setAudienceLevel(`*`audienceLevel`*`)`

- *audienceLevel*-a string containing the Audience Level.

  Important: The name of the *audienceLevel* must match the name of the audience level as defined in Campaign exactly. It is case-sensitive.

Return value

None.

Example

The following example is an excerpt from an `executeBatch` method calling `startSession` and `setAudience`.

```
String audienceLevel="Customer";
. . .
Command startSessionCommand = new CommandImpl();
startSessionCommand.setAudienceID(initialAudienceId);
. . .
Command setAudienceCommand = new CommandImpl();
setAudienceCommand.setAudienceLevel(audienceLevel);
. . .
/** Build command array */
Command[] commands =
    {
        startSessionCommand,
        setAudienceCommand,
    };
/** Make the call */
    BatchResponse batchResponse = api.executeBatch(sessionId, commands);

/** Process the response appropriately */
    processExecuteBatchResponse(batchResponse);
```

setDebug

The `setDebug` method defines the debug level for the `startSession` command.

`setDebug(`*`debug`*`)`

If `true`, the runtime server logs debug information to the runtime server log. If `false`, the runtime server does not log any debug information. The debug flag is set for each session individually. Therefore, you can trace debut data for an individual runtime session.

- **debug**-a boolean (`true` or `false`).

Return value

None.

Example

The following example is an excerpt from an executeBatch method calling startSession and setDebug.

```
boolean initialDebugFlag=true;
boolean newDebugFlag=false;
. . .
/* build the startSession command */
Command startSessionCommand = new CommandImpl();
startSessionCommand.setDebug(initialDebugFlag);
. . .

/* build the setDebug command */
Command setDebugCommand = new CommandImpl();
setDebugCommand.setMethodIdentifier(Command.COMMAND_SETDEBUG);
setDebugCommand.setDebug(newDebugFlag);

/** Build command array */
Command[] commands =
    {
        startSessionCommand,
        setDebugCommand,
    };
/** Make the call */
    BatchResponse batchResponse = api.executeBatch(sessionId, commands);

/** Process the response appropriately */
    processExecuteBatchResponse(batchResponse);
```

setEvent

The setEvent method defines the name of the event used by the postEvent command.

setEvent(*event*)

- **event**-A string which contains the event name.

Important: The name of the *event* must match the name of the event as defined in the interactive channel exactly. It is case-sensitive.

Return value

None.

Example

The following example is an excerpt from an executeBatch method calling postEvent.

```
String eventName = "SearchExecution";

Command postEventCommand = new CommandImpl();
postEventCommand.setMethodIdentifier(Command.COMMAND_POSTEVENT);
postEventCommand.setEventParameters(postEventParameters);
postEventCommand.setEvent(eventName);
```

setEventParameters

The setEventParameters method defines the event parameters used by the postEvent command. These values are stored in the session data.

setEventParameters(*eventParameters*)

- **eventParameters**-an array of NameValuePair objects defining the event parameters.

For example, if the event is logging an offer to contact history, you must include the treatment code of the offer.

Return value

None.

Example

The following example is an excerpt from an `executeBatch` method calling `postEvent`.

```
NameValuePair parmB1 = new NameValuePairImpl();
parmB1.setName("SearchString");
parmB1.setValueAsString("mortgage");
parmB1.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parmB2 = new NameValuePairImpl();
parmB2.setName("TimeStamp");
parmB2.setValueAsDate(new Date());
parmB2.setValueDataType(NameValuePair.DATA_TYPE_DATETIME);

NameValuePair parmB3 = new NameValuePairImpl();
parmB3.setName("Browser");
parmB3.setValueAsString("IE6");
parmB3.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parmB4 = new NameValuePairImpl();
parmB4.setName("FlashEnabled");
parmB4.setValueAsNumeric(1.0);
parmB4.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

NameValuePair parmB5 = new NameValuePairImpl();
parmB5.setName("TxAcctValueChange");
parmB5.setValueAsNumeric(0.0);
parmB5.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);

NameValuePair parmB6 = new NameValuePairImpl();
parmB6.setName("PageTopic");
parmB6.setValueAsString("");
parmB6.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair[] postEventParameters = { parmB1,
 parmB2,
 parmB3,
 parmB4,
 parmB5,
 parmB6
  };
. . .
Command postEventCommand = new CommandImpl();
postEventCommand.setMethodIdentifier(Command.COMMAND_POSTEVENT);
postEventCommand.setEventParameters(postEventParameters);
postEventCommand.setEvent(eventName);
```

setGetOfferRequests

The **setGetOfferRequests** method sets the parameter for retrieving offers used by the `getOffersForMultipleInteractionPoints` command.

`setGetOfferRequests(`*numberRequested*`)`

- **numberRequested** - an array of `GetOfferRequest` objects defining the parameter for retrieving offers.

Return value

None.

Example

The following example is an excerpt from a `GetOfferRequest` method calling `setGetOfferRequests`.

```
GetOfferRequest request1 = new GetOfferRequest(5, GetOfferRequest.NO_DUPLICATION);
     request1.setIpName("IP1");
     OfferAttributeRequirements offerAttributes1 = new OfferAttributeRequirements();
     NameValuePairImpl attr1 = new NameValuePairImpl("attr1",
          NameValuePair.DATA_TYPE_NUMERIC, 1);
     NameValuePairImpl attr2 = new NameValuePairImpl("attr2",
          NameValuePair.DATA_TYPE_STRING, "value2");
     NameValuePairImpl attr3 = new NameValuePairImpl("attr3",
          NameValuePair.DATA_TYPE_STRING, "value3");
     NameValuePairImpl attr4 = new NameValuePairImpl("attr4",
          NameValuePair.DATA_TYPE_NUMERIC, 4);
     offerAttributes1.setNumberRequested(5);
     offerAttributes1.setAttributes(new NameValuePairImpl[] {attr1, attr2});
     OfferAttributeRequirements childAttributes1 = new OfferAttributeRequirements();
     childAttributes1.setNumberRequested(3);
     childAttributes1.setAttributes(new NameValuePairImpl[] {attr3});
     OfferAttributeRequirements childAttributes2 = new OfferAttributeRequirements();
     childAttributes2.setNumberRequested(3);
     childAttributes2.setAttributes(new NameValuePairImpl[] {attr4});
     offerAttributes1.setChildRequirements(Arrays.asList(childAttributes1,
          childAttributes2));
     request1.setOfferAttributes(offerAttributes1);

     GetOfferRequest request2 = new GetOfferRequest(3, GetOfferRequest.ALLOW_DUPLICATION);
     request2.setIpName("IP2");
     OfferAttributeRequirements offerAttributes2 = new OfferAttributeRequirements();
     offerAttributes2.setNumberRequested(3);
     offerAttributes2.setAttributes(new NameValuePairImpl[] {new NameValuePairImpl("attr5",
          NameValuePair.DATA_TYPE_STRING, "value5")});
     request2.setOfferAttributes(offerAttributes2);

     GetOfferRequest request3 = new GetOfferRequest(2, GetOfferRequest.NO_DUPLICATION);
     request3.setIpName("IP3");
     request3.setOfferAttributes(null);

     Command getOffersMultiIPCmd = new CommandImpl();
     getOffersMultiIPCmd.setGetOfferRequests(new GetOfferRequest[] {request1,
          request2, request3});
```

setInteractiveChannel

The `setInteractiveChannel` method defines the name of the interactive channel used by the `startSession` command.

`setInteractiveChannel(`*`interactiveChannel`*`)`

- **interactiveChannel**-a string containing the interactive channel name.

Important: The *interactiveChannel* must match the name of the interactive channel as defined in Campaign exactly. It is case-sensitive.

Return value

None.

Example

The following example is an excerpt from an `executeBatch` method calling `startSession`.

```
String interactiveChannel="Accounts Website";
. . .
Command startSessionCommand = new CommandImpl();
startSessionCommand.setInteractiveChannel(interactiveChannel);
```

setInteractionPoint

The `setInteractionPoint` method defines the name of the interaction point used by the `getOffers` and `postEvent` commands.

`setInteractionPoint(`*`interactionPoint`*`)`

- **interactionPoint**-a string containing the interaction point name.

Important: The *interactionPoint* must match the name of the interaction point as defined in the interactive channel exactly. It is case-sensitive.

Return value

None.

Example

The following example is an excerpt from an `executeBatch` method calling `getOffers`.

```
String interactionPoint = "Overview Page Banner 1";
int numberRequested=1;

Command getOffersCommand = new CommandImpl();
getOffersCommand.setMethodIdentifier(Command.COMMAND_GETOFFERS);
getOffersCommand.setInteractionPoint(interactionPoint);
getOffersCommand.setNumberRequested(numberRequested);
```

setMethodIdentifier

The `setMethodIdentifier` method defines the type of command contained in the command object.

`setMethodIdentifier(`*`methodIdentifier`*`)`

- **methodIdentifier**-a string containing the type of command.

  The valid values are:

  - **COMMAND_ENDSESSION**-represents the `endSession` method.
  - **COMMAND_GETOFFERS**-represents the `getOffers` method.
  - **COMMAND_GETPROFILE**-represents the `getProfile` method.
  - **COMMAND_GETVERSION**-represents the `getVersion` method.
  - **COMMAND_POSTEVENT**-represents the `postEvent` method.
  - **COMMAND_SETAUDIENCE**-represents the `setAudience` method.
  - **COMMAND_SETDEBUG**-represents the `setDebug` method.
  - **COMMAND_STARTSESSION**-represents the `startSession` method.

Return value

None.

Example

The following example is an excerpt from an executeBatch method calling getVersion and endSession.

```
Command getVersionCommand = new CommandImpl();
getVersionCommand.setMethodIdentifier(Command.COMMAND_GETVERSION);

Command endSessionCommand = new CommandImpl();
endSessionCommand.setMethodIdentifier(Command.COMMAND_ENDSESSION);

Command[] commands =
{
    getVersionCommand,
    endSessionCommand
};
```

setNumberRequested

The setNumberRequested method defines the number of offers requested by the getOffers command.

setNumberRequested(*numberRequested*)

- **numberRequested**-an integer defining the number of offers requested by the getOffers command.

Return value

None.

Example

The following example is an excerpt from an executeBatch method calling getOffers.

```
String interactionPoint = "Overview Page Banner 1";
int numberRequested=1;

Command getOffersCommand = new CommandImpl();
getOffersCommand.setMethodIdentifier(Command.COMMAND_GETOFFERS);
getOffersCommand.setInteractionPoint(interactionPoint);
getOffersCommand.setNumberRequested(numberRequested);
```

setRelyOnExistingSession

The setRelyOnExistingSession method defines a boolean defining whether the startSession command uses an existing session or not.

setRelyOnExistingSession(*relyOnExistingSession*)

If true, the session ID for executeBatch must match an existing session ID. If false, you must supply a new session ID with the executeBatch method.

- **relyOnExistingSession**-a boolean (true or false).

Return value

None.

Example

The following example is an excerpt from an executeBatch method calling startSession.

```
boolean relyOnExistingSession=false;
. . .
Command startSessionCommand = new CommandImpl();
startSessionCommand.setRelyOnExistingSession(relyOnExistingSession);
```

About the NameValuePair interface

Many methods in the Interact API either return NameValuePair objects or require you to pass NameValuePair objects as arguments. When passing as arguments into a method, you should use the default implementation NameValuePairImpl.

getName

The getName method returns the name component of a NameValuePair object.

```
getName()
```

Return value

The getName method returns a string.

Example

The following example is an excerpt from a method which processes the response object for getProfile.

```
for(NameValuePair nvp : response.getProfileRecord())
{
    System.out.println("Name:"+nvp.getName());
}
```

getValueAsDate

The getValueAsDate method returns the value of a NameValuePair object.

```
getValueAsDate()
```

You should use getValueDataType before using getValueAsDate to confirm you are referencing the correct data type.

Return value

The getValueAsDate method returns a date.

Example

The following example is an excerpt from a method which processes a NameValuePair and prints the value if it is a date.

```
if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_DATE))
{
    System.out.println("Value:"+nvp.getValueAsDate());
}
```

getValueAsNumeric

The getValueAsNumeric method returns the value of a NameValuePair object.

```
getValueAsNumeric()
```

You should use getValueDataType before using getValueAsNumeric to confirm you are referencing the correct data type.

Return value

The getValueAsNumeric method returns a double. If, for example, you are retrieving a value originally stored in your profile table as an Integer, getValueAsNumeric returns a double.

Example

The following example is an excerpt from a method which processes a NameValuePair and prints the value if it is numeric.

```
if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_NUMERIC))
{
    System.out.println("Value:"+nvp.getValueAsNumeric());
}
```

getValueAsString

The getValueAsString method returns the value of a NameValuePair object.

```
getValueAsString()
```

You should use getValueDataType before using getValueAsString to confirm you are referencing the correct data type.

Return value

The getValueAsString method returns a string. If, for example, you are retrieving a value originally stored in your profile table as a char, varchar, or char[10], getValueAsString returns a string.

Example

The following example is an excerpt from a method which processes a NameValuePair and prints the value if it is a string.

```
if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_STRING))
{
    System.out.println("Value:"+nvp.getValueAsString());
}
```

getValueDataType

The getValueDataType method returns the data type of a NameValuePair object.

```
getValueDataType()
```

You should use getValueDataType before using getValueAsDate, getValueAsNumeric, or getValueAsString to confirm you are referencing the correct data type.

Return value

The getValueDataType method returns a string indicating whether the NameValuePair contains a data, number, or string.

The valid values are:

- **DATA_TYPE_DATETIME**-a date containing a date and time value.

- **DATA_TYPE_NUMERIC**-a double containing a number value.
- **DATA_TYPE_STRING**-a string containing a text value.

Example

The following example is an excerpt from a method which processes the response object from a `getProfile` method.

```
for(NameValuePair nvp : response.getProfileRecord())
{
    System.out.println("Name:"+nvp.getName());
    if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_DATETIME))
    {
        System.out.println("Value:"+nvp.getValueAsDate());
    }
    else if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_NUMERIC))
    {
        System.out.println("Value:"+nvp.getValueAsNumeric());
    }
    else
    {
        System.out.println("Value:"+nvp.getValueAsString());
    }
}
```

setName

The `setName` method defines the name component of a NameValuePair object.

`setName(`*`name`*`)`

- **name**-a string containing the name component of a NameValuePair object.

Return value

None.

Example

The following example shows how to define the name component of a NameValuePair.

```
NameValuePair custId = new NameValuePairImpl();
custId.setName("CustomerId");
custId.setValueAsNumeric(1.0);
custId.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
NameValuePair[] initialAudienceId = { custId };
```

setValueAsDate

The `setValueAsDate` method defines the value of a NameValuePair object.

`setValueAsDate(`*`valueAsDate`*`)`

- **valueAsDate**-a date containing the date and time value of a NameValuePair object.

Return value

None.

Example

The following example shows how to define the value component of a NameValuePair if the value is a date.

```
NameValuePair parm2 = new NameValuePairImpl();
parm2.setName("TimeStamp");
parm2.setValueAsDate(new Date());
parm2.setValueDataType(NameValuePair.DATA_TYPE_DATETIME);
```

setValueAsNumeric

The `setValueAsNumeric` method defines the value of a NameValuePair object.

`setValueAsNumeric(`*`valueAsNumeric`*`)`

- **valueAsNumeric**-a double containing the numeric value of a NameValuePair object.

Return value

None.

Example

The following example shows how to define the value component of a NameValuePair if the value is a numeric.

```
NameValuePair parm4 = new NameValuePairImpl();
parm4.setName("FlashEnabled");
parm4.setValueAsNumeric(1.0);
parm4.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
```

setValueAsString

The `setValueAsString` method defines the value of a NameValuePair object.

`setValueAsString(`*`valueAsString`*`)`

- **valueAsString**-a string containing the value of a NameValuePair object

Return value

None.

Example

The following example shows how to define the value component of a NameValuePair if the value is a numeric.

```
NameValuePair parm3 = new NameValuePairImpl();
parm3.setName("Browser");
parm3.setValueAsString("IE6");
parm3.setValueDataType(NameValuePair.DATA_TYPE_STRING);
```

setValueDataType

The `setValueDataType` method defines the data type of a NameValuePair object.

`getValueDataType(`*`valueDataType`*`)`

The valid values are:

- **DATA_TYPE_DATETIME**-a date containing a date and time value.
- **DATA_TYPE_NUMERIC**-a double containing a number value.
- **DATA_TYPE_STRING**-a string containing a text value.

Return value

None.

Example

The following examples show how to set the data type of the value of a NameValuePair.

```
NameValuePair parm2 = new NameValuePairImpl();
parm2.setName("TimeStamp");
parm2.setValueAsDate(new Date());
parm2.setValueDataType(NameValuePair.DATA_TYPE_DATETIME);

NameValuePair parm3 = new NameValuePairImpl();
parm3.setName("Browser");
parm3.setValueAsString("IE6");
parm3.setValueDataType(NameValuePair.DATA_TYPE_STRING);

NameValuePair parm4 = new NameValuePairImpl();
parm4.setName("FlashEnabled");
parm4.setValueAsNumeric(1.0);
parm4.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
```

About the Offer class

The `Offer` class contains methods which define an Offer object. This offer object contains many of the same properties of an offer in Campaign.

The offer object contains the following attributes:

- **AdditionalAttributes**-NameValuePairs containing any custom offer attributes you have defined in Campaign.
- **Description**-The description of the offer.
- **EffectiveDate**-The effective date of the offer.
- **ExpirationDate**-The expiration date of the offer.
- **OfferCode**-The offer code of the offer.
- **OfferName**-The name of the offer.
- **TreatmentCode**-The treatment code of the offer.
- **Score**-The marketing score of the offer, or the score defined by the `ScoreOverrideTable` if the `enableScoreOverrideLookup` property is true.

getAdditionalAttributes

The `getAdditionalAttributes` method returns the custom offer attributes defined in Campaign.

```
getAdditionalAttributes()
```

Return value

The `getAdditionalAttributes` method returns an array of NameValuePair objects.

Example

The following example sorts through all the additional attributes, checking for the effective date and expiration date, and printing out the other attributes.

```
for(NameValuePair offerAttribute : offer.getAdditionalAttributes())
{
    // check to see if the effective date exists
    if(offerAttribute.getName().equalsIgnoreCase("effectiveDate"))
    {
        System.out.println("Found effective date");
    }
```

```
        // check to see if the expiration date exists
        else if(offerAttribute.getName().equalsIgnoreCase("expirationDate"))
        {
            System.out.println("Found expiration date");
        }
        printNameValuePair(offerAttribute);
        }
    }
    public static void printNameValuePair(NameValuePair nvp)
    {
        // print out the name:
        System.out.println("Name:"+nvp.getName());

        // based on the datatype, call the appropriate method to get the value
        if(nvp.getValueDataType()==NameValuePair.DATA_TYPE_DATETIME)
            System.out.println("DateValue:"+nvp.getValueAsDate());
        else if(nvp.getValueDataType()==NameValuePair.DATA_TYPE_NUMERIC)
            System.out.println("NumericValue:"+nvp.getValueAsNumeric());
        else
            System.out.println("StringValue:"+nvp.getValueAsString());
    }
```

getDescription

The `getDescription` method returns the description of the offer defined in Campaign.

```
getDescription()
```

Return value

The `getDescription` method returns a string.

Example

The following example prints the description of an offer.

```
for(Offer offer : offerList.getRecommendedOffers())
{
    // print offer
     System.out.println("Offer Description:"+offer.getDescription());
}
```

getOfferCode

The `getOfferCode` method returns the offer code of the offer as defined in Campaign.

```
getOfferCode()
```

Return value

The `getOfferCode` method returns an array of strings containing the offer code of the offer.

Example

The following example prints the offer code of an offer.

```
for(Offer offer : offerList.getRecommendedOffers())
{
    // print offer
   System.out.println("Offer Code:"+offer.getOfferCode());
}
```

getOfferName

The `getOfferName` method returns the name of the offer as defined in Campaign.

```
getOfferName()
```

Return value

The `getOfferName` method returns string.

Example

The following example prints the name of an offer.

```
for(Offer offer : offerList.getRecommendedOffers())
{
// print offer
System.out.println("Offer Name:"+offer.getOfferName());
}
```

getScore

The `getScore` method returns a score, which is based on the offers you configured.

```
getScore()
```

The `getScore` method returns one of the following:

- If you did not enabled the default offers table, score override table, or built-in learning, this method returns the marketing score of the offer as defined on the interaction strategy tab.
- If you enabled the default offers or score override table and not enabled built-in learning, this method returns the score of the offer as defined by the order of precedence between the default offers table, the marketer's score, and the score override table.
- If you enabled built-in learning, this method returns the final score that the built-in learning used to order offers.

Return value

The `getScore` method returns an integer that represents the score of the offer.

Example

The following example prints the score of an offer.

```
for(Offer offer : offerList.getRecommendedOffers())
{
// print offer
System.out.println("Offer Score:"+offer.getOfferScore());
}
```

getTreatmentCode

The `getTreatmentCode` method returns the treatment code of the offer as defined in Campaign.

```
getTreatmentCode()
```

Because Campaign uses the treatment code to identify the instance of the offer served, this code must be returned as an event parameter when using the `postEvent` method to log a contact, acceptance, or rejection event of the offer. If

you are logging an offer acceptance or rejection, you must set the name value of the NameValuePair representing the treatment code to `UACIOfferTrackingCode`.

Return value

The `getTreatmentCode` method returns a string.

Example

The following example prints the treatment code of an offer.

```
for(Offer offer : offerList.getRecommendedOffers())
{
    // print offer
    System.out.println("Offer Treatment Code:"+offer.getTreatmentCode());
}
```

About the OfferList class

The `OfferList` class contains methods which define the results of the `getOffers` method.

The OfferList object contains the following attributes:

- **DefaultString**-The default string defined for the interaction point in the interactive channel.
- **RecommendedOffers**-An array of the Offer objects requested by the `getOffers` method.

The `OfferList` class works with lists of offers. This class is not related to Campaign offer lists.

getDefaultString

The `getDefaultString` method returns the default string for the interaction point as defined in Campaign.

```
getDefaultString()
```

If the RecommendedOffers object is empty, you should configure your touchpoint to present this string to ensure some content is presented. Interact populates the DefaultString object only if the RecommendedOffers object is empty.

Return value

The `getDefaultString` method returns a string.

Example

The following example gets the default string if the offerList object does not contain any offers.

```
OfferList offerList=response.getOfferList();
if(offerList.getRecommendedOffers() != null)
{
    for(Offer offer : offerList.getRecommendedOffers())
    {
  System.out.println("Offer Name:"+offer.getOfferName());
    }
}
else // count on the default Offer String
    System.out.println("Default offer:"+offerList.getDefaultString());
```

getRecommendedOffers

The getRecommendedOffers method returns an array of Offer objects requested by the getOffers method.

```
getRecommendedOffers()
```

If the response to getRecommendedOffer is empty, the touchpoint should present the result of getDefaultString.

Return value

The getRecommendedOffers method returns an Offer object.

Example

The following example processes the OfferList object, and prints the offer name for all the recommended offers.

```
OfferList offerList=response.getOfferList();
if(offerList.getRecommendedOffers() != null)
{
    for(Offer offer : offerList.getRecommendedOffers())
    {
        // print offer
        System.out.println("Offer Name:"+offer.getOfferName());
    }
}
else // count on the default Offer String
System.out.println("Default offer:"+offerList.getDefaultString());
```

About the Response class

The Response class contains methods which define the results of any of the InteractAPI class methods.

The Response object contains the following attributes:

- **AdvisoryMessages**-an array of advisory messages. This attribute is populated only if there were warnings or errors when the method ran.
- **ApiVersion**-a string containing the API version. This attribute is populated by the getVersion method.
- **OfferList**-the OfferList object containing the offers requested by the getOffers method.
- **ProfileRecord**-an array of NameValuePairs containing profile data. This attribute is populated by the getProfile method.
- **SessionID**-a string defining the session ID. This is returned by all InteractAPI class methods.
- **StatusCode**-a number stating if the method ran without error, with a warning, or with errors. This is returned by all InteractAPI class methods.

getAdvisoryMessages

The getAdvisoryMessages method returns an array of Advisory Messages from the Response object.

```
getAdvisoryMessages()
```

Return value

The getAdvisoryMessages method returns an array of Advisory Message objects.

Example

The following example gets the AdvisoryMessage objects from a Response object and iterates through them, printing out the messages.

```
AdvisoryMessage[] messages = response.getAdvisoryMessages();
    for(AdvisoryMessage msg : messages)
    {
        System.out.println(msg.getMessage());
        // Some advisory messages may have additional detail:
        System.out.println(msg.getDetailMessage());
    }
```

getApiVersion

The `getApiVersion` method returns the API version of a Response object.

```
getApiVersion()
```

The `getVersion` method populates the ApiVersion attribute of a Response object.

Return value

The Response object returns a string.

Example

The following example is an excerpt from a method which processes the response object for `getVersion`.

```
if(response.getStatusCode() == Response.STATUS_SUCCESS)
{
    System.out.println("getVersion call processed with no warnings or errors");
    System.out.println("API Version:" + response.getApiVersion());
}
```

getOfferList

The `getOfferList` method returns the OfferList object of a Response object.

```
getOfferList()
```

The `getOffers` method populates the OfferList object of a Response object.

Return value

The Response object returns an OfferList object.

Example

The following example is an excerpt from a method which processes the response object for `getOffers`.

```
OfferList offerList=response.getOfferList();
if(offerList.getRecommendedOffers() != null)
{
    for(Offer offer : offerList.getRecommendedOffers())
    {
    // print offer
    System.out.println("Offer Name:"+offer.getOfferName());
    }
}
```

getAllOfferLists

The `getAllOfferLists` method returns an array of all OfferLists of a Response object.

```
getAllOfferLists()
```

This is used by the `getOffersForMultipleInteractionPoints` method that populates the OfferList array object of a Response object.

Return value

The Response object returns an OfferList array.

Example

The following example is an excerpt from a method which processes the response object for `getOffers`.

```
OfferList[] allOfferLists = response.getAllOfferLists();
 if (allOfferLists != null) {
  for (OfferList ol : allOfferLists) {
   System.out.println("The following offers are delivered for interaction point "
     + ol.getInteractionPointName() + ":");
   for (Offer o : ol.getRecommendedOffers()) {
    System.out.println(o.getOfferName());
   }
  }
 }
```

getProfileRecord

The `getProfileRecord` method returns the profile records for the current session as an array of NameValuePair objects. These profile records also include any `eventParameters` added earlier in the runtime session.

```
getProfileRecord()
```

The `getProfile` method populates the profile record NameValuePair objects of a Response object.

Return value

The Response object returns an array of NameValuePair objects.

Example

The following example is an excerpt from a method which processes the response object for `getOffers`.

```
for(NameValuePair nvp : response.getProfileRecord())
{
    System.out.println("Name:"+nvp.getName());
    if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_DATETIME))
    {
        System.out.println("Value:"+nvp.getValueAsDate());
    }
    else if(nvp.getValueDataType().equals(NameValuePair.DATA_TYPE_NUMERIC))
    {
         System.out.println("Value:"+nvp.getValueAsNumeric());
    }
    else
```

```
    {
        System.out.println("Value:"+nvp.getValueAsString());
    }
}
```

getSessionID

The `getSessionID` method returns session ID.

```
getSessionID()
```

Return value

The `getSessionID` method returns a string.

Example

The following example shows a message you can display at the end or beginning of your error handling to indicate to which session any errors pertain.

```
System.out.println("This response pertains to sessionId:"+response.getSessionID());
```

getStatusCode

The `getStatusCode` method returns the status code of a Response object.

```
getStatusCode()
```

Return value

The Response object returns an integer.

- 0 - STATUS_SUCCESS - The method called completed with no errors. There may or may not be Advisory Messages.
- 1 - STATUS_WARNING - The method called completed with at least one warning message (but no errors). Query Advisory Messages for more details.
- 2 - STATUS_ERROR - The method called did not complete successfully and has at least one error message. Query Advisory Messages for more details.

Example

The following is an example of how you can use `getStatusCode` in error handling.

```
public static void processSetDebugResponse(Response response)
{
    // check if response is successful or not
    if(response.getStatusCode() == Response.STATUS_SUCCESS)
    {
        System.out.println("setDebug call processed with no warnings or errors");
    }
    else if(response.getStatusCode() == Response.STATUS_WARNING)
    {
        System.out.println("setDebug call processed with a warning");
    }
    else
    {
        System.out.println("setDebug call processed with an error");
    }

    // For any non-successes, there should be advisory messages explaining why
    if(response.getStatusCode() != Response.STATUS_SUCCESS)
        printDetailMessageOfWarningOrError("setDebug",
    response.getAdvisoryMessages());
}
```

Chapter 8. Classes and methods for the IBM Interact JavaScript API

The following sections list requirements and other details you should know before you work with the Interact JavaScript API.

The Interact API supports a javascript flavor to allow for end-user client (browser) to server communication.

Note: This section assumes you are familiar with a JavaScript-based API.

Note: Multiple occurrences of any parameter in a single API call is not supported.

JavaScript prerequisites

Before you use the Interact JavaScript API on a website you must include the `interactapi.js` file on your web pages.

Working with session data

When you initiate a session with the `startSession` method, session data is loaded into memory. Throughout the session, you can read and write to the session data (which is a superset of the static profile data).

The session contains the following data:

- Static profile data
- Segment assignments
- Real-time data
- Offer recommendations

All session data is available until you call the `endSession` method, or the `sessionTimeout` time elapses. Once the session ends, all data not explicitly saved to contact or response history or some other database table is lost.

The data is stored as a set of name-value pairs. If the data is read from a database table, the name is the column of the table.

You can create these name-value pairs as you work with the Interact API. You do not need to declare all name-value pairs in a global area. If you set new event parameters as name-value pairs, the runtime environment adds the name-value pairs to the session data. For example if you use event parameters with the `postEvent` method, the runtime environment adds the event parameters to the session data, even if the event parameters were not available in the profile data. This data exists in the session data only.

You can overwrite session data at any time. For example, if part of the customer profile includes `creditScore`, you can pass in an event parameter using the custom type `NameValuePair`. In the `NameValuePair` class, you can use the `setName` and `setValueAsNumeric` methods to change the value. The name needs to match. Within the session data, the name is not case-sensitive. Therefore, the name `creditscore` or `CrEdItScOrE` would both overwrite `creditScore`.

Only the last data written to the session data is kept. For example, `startSession` loads the profile data for the value of `lastOffer`. A `postEvent` method overwrites `lastOffer`. Then a second `postEvent` method overwrites `lastOffer`. The runtime environment keeps only the data written by the second `postEvent` method in the session data.

When the session ends, the data is lost, unless you made special considerations such as using a Snapshot process in your interactive flowchart to write the data to a database table. If you are planning on using Snapshot processes, remember that the names need to match the limitations of your database. For example, if your are allowed only 256 characters for the name of a column, then the name for the name-value pair should not exceed 256 characters.

Working with the callback parameter

The callback function is an additional parameter of each method of the Interact JavaScript API.

The main browser process is a single threaded event loop. Executing a long-running operation within a single-threaded event loop, blocks the process. This is stops the process from processing other events while it waits for your operation to complete. In order to prevent blocks on long-running operations, the XMLHttpRequest provides an asynchronous interface. You pass it a callback to run after the operation is complete, and while it processes, it gives control back to the main event loop instead of blocking the process.

If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

For example, if you wanted to display offers on your web page, you would use the `getOffers` method and use the callback to display on the page. The web page behaves normally and does not wait for Interact to return the offers. Instead, when Interact does return the offers, the response is sent back in the callback parameter. You can parse the callback data and show offers on the page.

You can use one generic callback for all functions or you can also use specific callbacks for specific functions.

You can use `var callback = InteractAPI.Callback.create(onSuccess, onError);` to create a generic callback function.

You can use the following function to create a specific callback function for the getOffers method.

```
var callbackforGetOffer = InteractAPI.Callback.create(onSuccessofGetOffer,
onErrorofGetOffer);
```

About the InteractAPI class

The InteractAPI class contains the methods which you use to integrate your touchpoint with the runtime server. All other classes and methods in the Interact API support the methods in this class.

You must compile your implementation against `interact_client.jar` located in the `lib` directory of your Interact runtime environment installation.

startSession

The startSession method creates and defines a runtime session.

```
function callStartSession(commandsToExecute, callback) {

        //read configured start session
        var ssId = document.getElementById('ss_sessionId').value;
        var icName = document.getElementById('ic').value;
        var audId = document.getElementById('audienceId').value;
        var audLevel = document.getElementById('audienceLevel').value;
        var params = document.getElementById('ss_parameters').value;
        var relyOldSs = document.getElementById('relyOnOldSession').value;
        var debug = document.getElementById('ss_isDebug').value;

        InteractAPI.startSession(ssId, icName,
                                    getNameValuePairs(audId), audLevel,
                                    getNameValuePairs(params), relyOldSs,
                                    debug, callback) ;

}
```

startSession can trigger up to five actions:

- create a runtime session.
- load visitor profile data for the current audience level into the runtime session, including any dimension tables marked for loading in the table mapping defined for the interactive channel.
- trigger segmentation, running all interactive flowcharts for the current audience level.
- load offer suppression data into the session, if the enableOfferSuppressionLookup property is set to true.
- load score override data into the session, if the enableScoreOverrideLookup property is set to true.

The startSession method requires the following parameters:

- **sessionID**-a string which identifies the session ID. You must define the session ID. For example, you could use a combination of customer ID and timestamp.

  To define what makes a runtime session, a session id has to be specified. This value is managed by the client. All method calls for the same session id has to be synchronized by the client. The behavior for concurrent API calls with the same session id is undefined.
- **relyOnExistingSession** - a boolean which defines whether this session uses a new or an existing session. Valid values are true or false. If true, you must supply an existing session ID with the startSession method. If false, you must supply a new session ID.

  If you set relyOnExistingSession to true and a session exists, the runtime environment uses the existing session data and does not reload any data or trigger segmentation. If the session does not exist, the runtime environment creates a new session, including loading data and triggering segmentation. Setting relyOnExistingSession to true and using it with all startSession calls is useful if your touchpoint has a longer session length than the runtime session. For example, a web site session is alive for 2 hours, but the runtime session is only alive for 20 minutes.

  If you call startSession twice with the same session ID, all session data from the first startSession call is lost if relyOnExistingSession is false.

- **debug** - a boolean which enables or disables debug information. Valid values are `true` or `false`. If true, Interact logs debug information to the runtime server logs. The debug flag is set for each session individually. Therefore, you can trace debug data for an individual session.
- **interactiveChannel**-a string defining the name of the interactive channel this session refers to. This name must match the name of the interactive channel defined in Campaign exactly.
- **audienceID** - an array of NameValuePairImpl objects where the names must match the physical column names of any table containing the audience ID.
- **audienceLevel** - a string defining the audience level.
- **parameters** - NameValuePairImpl objects identifying any parameters that need to be passed with `startSession`. These values are stored in the session data and can be used for segmentation.

  If you have several interactive flowcharts for the same audience level, you must include a superset of all columns in all the tables. If you configure the runtime to load the profile table, and the profile table contains all the columns you require, you do not need to pass any parameters, unless you want to overwrite the data in the profile table. If your profile table contains a subset of the required columns, you must include the missing columns as parameters.
- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

If the `audienceID` or `audienceLevel` are invalid and `relyOnExistingSession` is false, the `startSession` call fails. If the `interactiveChannel` is invalid, `startSession` fails, whether `relyOnExistingSession` is true or false.

If `relyOnExistingSession` is true, and you make a second `startSession` call using the same `sessionID`, but the first session has expired, Interact creates a new session.

If `relyOnExistingSession` is true, and you make a second `startSession` call using the same `sessionID` but a different `audienceID` or `audienceLevel`, the runtime server changes the audience for the existing session.

If `relyOnExistingSession` is true, and you make a second `startSession` call using the same `sessionID` but a different `interactiveChannel`, the runtime server creates a new session.

Return value

The runtime server responds to `startSession` with a Response object with the following attributes populated:

- AdvisoryMessages (if StatusCode does not equal 0)
- ApiVersion
- SessionID
- StatusCode

Offer deduplication across offer attributes

Using the Interact application programming interface (API), two API calls deliver offers: `getOffers` and `getOffersForMultipleInteractionPoints`. `getOffersForMultipleInteractionPoints` can prevent the return of duplicate offers at the *OfferID* level, but cannot deduplicate offers across offer category. So, for example, for Interact to return only one offer from each offer category, a

workaround was previously required. With the introduction of two parameters to the `startSession` API call, offer deduplication across offer attributes, such as category, is now possible.

This list summarizes the parameters that were added to the `startSession` API call. For more information about these parameters or any aspect of the Interact API, see the *IBM Interact Administrator's Guide*, or the Javadoc files included with your Interact installation in `<Interact_Home>/docs/apiJavaDoc`.

- `UACIOfferDedupeAttribute`. To create a `startSession` API call with offer deduplication, so that the subsequent `getOffer` calls return only one offer from each category, you must include the `UACIOfferDedupeAttribute` parameter as part of the API call. You can specify a parameter in the `name,value,type` format, as shown here:

  `UACIOfferDedupeAttribute,<attributeName>,string`

  In this example, you would replace *<attributeName>* with the name of the offer attribute you want to use as the criterion for deduplication, such as Category.

  Note: Interact examines the offers that have the same attribute value you specify (such as Category) and deduplicate to remove all but the offer that has the highest score. If the offers that have the duplicate attribute also have identical scores, Interact returns a random selection from among the matching offers.

- `UACINoAttributeDedupeIfFewerOffer`. When you include the `UACIOfferDedupeAttribute` in the `startSession` call, you can also set this `UACINoAttributeDedupeIfFewerOffer` parameter to specify the behavior in cases where the offer list after deduplication no longer contains enough offers to satisfy the original request.

  For example, if you set `UACIOfferDedupeAttribute` to use the offer category to deduplicate offers, and your subsequent getOffers call requests that eight offers be returned, deduplication might result in fewer than eight eligible offers. In that case, setting `UACINoAttributeDedupeIfFewerOffer` parameter to true would result in adding some of the duplicated to the eligible list to satisfy the requested number of offers. In this example, if you set the parameter to false, the number of offers that are returned would be fewer than the requested number.

  `UACINoAttributeDedupeIfFewerOffer` is set to true by default.

For example, suppose you specified as a `startSession` parameter that the deduplication criterion is offer Category, as shown here:

```
UACIOfferDedupeAttribute,Category,string;

UACINoAttributeDedupeIfFewerOffer,1,string
```

By default, the `UACIOfferDedupeAttribute` will not deduplicate offers if fewer than the requested number is returned . However, to ensure that the deduplication happens when fewer than requested offers are returned, the `UACINoAttributeDedupeIfFewerOffer` parameter must be provided and must be set to 1.

These parameters together cause Interact to deduplicate offers based on the offer attribute "Category," and to return only the deduplicated offers even if the resulting number of offers is fewer than requested (`UACINoAttributeDedupeIfFewerOffer` is false).

When you issue a getOffers API call, the original set of eligible offers might include these offers:

- Category=Electronics: Offer A1 with a score of 100 and Offer A2 with a score of 50.
- Category=Smartphones: Offer B1 with a score of 100, Offer B2 with a score of 80, and offer B3 with a score of 50.
- Category=MP3Players: Offer C1 with a score of 100, Offer C2 with a score of 50.

In this case, there were two duplicate offers that match the first category, three duplicate offers that match the second category, and two duplicate offers that match the third category. The offers that are returned would be the highest scoring offers from each category, which are Offer A1, Offer B1, and Offer C1.

If the getOffers API call requested six offers, this example set UACINoAttributeDedupeIfFewerOffer to false, so only three offers would be returned.

If the getOffers API call requested six offers, and this example omitted the UACINoAttributeDedupeIfFewerOffer parameter, or specifically set it to true, some of the duplicate offers would be included in the result to satisfy the requested number.

postEvent

The postEvent method enables you to execute any event defined in the interactive channel.

```
function callPostEvent(commandsToExecute, callback) {

        var ssId = document.getElementById('pe_sessionId').value;
        var ev = document.getElementById('event').value;
        var params = document.getElementById('parameters').value;

        InteractAPI.postEvent(ssId, ev, getNameValuePairs(params), callback);

}
```

- **sessionID**: a string identifying the session ID.
- **eventName**: a string identifying the name of the event.

  Note: The name of the event must match the name of the event as defined in the interactive channel. This name is case-insensitive.
- **eventParameters**. NameValuePairImpl objects identifying any parameters that need to be passed with the event. These values are stored in the session data.

  If this event triggers re-segmentation, you must ensure that all data required by the interactive flowcharts is available in the session data. If any of these values have not been populated by prior actions (for example, from startSession or setAudience, or loading the profile table) you must include an eventParameter for every missing value. For example, if you have configured all profile tables to load into memory, you must include a NameValuePair for temporal data required for the interactive flowcharts.

  If you are using more than one audience level, you most likely have different sets of eventParameters for each audience level. You should include some logic to ensure you are selecting the correct set of parameters for the audience level.

  Important: If this event logs to response history, you must pass the treatment code for the offer. You must define the name for the NameValuePair as "UACIOfferTrackingCode".

You can only pass one treatment code per event. If you do not pass the treatment code for an offer contact, Interact logs an offer contact for every offer in the last recommended list of offers. If you do not pass the treatment code for a response, Interact returns an error.

- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.
- There are several other reserved parameters used with `postEvent` and other methods and are discussed later in this section.

Any request for re-segmentation or writing to contact or response history does not wait for a response.

Re-segmentation does not clear prior segmentation results for the current audience level. You can use the `UACIExecuteFlowchartByName` parameter to define specific flowcharts to run. The `getOffers` method waits for re-segmentation to finish before running. Therefore, if you call a `postEvent` method, which triggers a re-segmentation immediately before a `getOffers` call, there might be a delay.

Return value

The runtime server responds to `postEvent` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList
- Profile
- SessionID
- StatusCode

getOffers

The `getOffers` method enables you to request offers from the runtime server.

```
function callGetOffers(commandsToExecute, callback) {

        var ssId = document.getElementById('go_sessionId').value;
        var ip = document.getElementById('go_ipoint').value;
        var nofRequested = 5 ;
        var nreqString = document.getElementById('offersRequested').value;

        InteractAPI.getOffers(ssId, ip, nofRequested, callback);

}
```

- **session ID**-a string identifying the current session.
- **Interaction point**-a string identifying the name of the interaction point this method references.

  Note: This name must match the name of the interaction point defined in interactive channel exactly.
- **nofRequested**-an integer identifying the number of offers requested.
- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

The `getOffers` method waits the number of milliseconds defined in the `segmentationMaxWaitTimeInMS` property for all re-segmentation to complete before

running. Therefore, if you call a `postEvent` method which triggers a re-segmentation or a `setAudience` method immediately before a `getOffers` call, there may be a delay.

Return value

The runtime server responds to `getOffers` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList
- Profile
- SessionID
- StatusCode

getOffersForMultipleInteractionPoints

The `getOffersForMultipleInteractionPoints` method enables you to request offers from the runtime server for multiple IPs with deduplication.

```
function callGetOffersForMultipleInteractionPoints(commandsToExecute, callback) {

        var ssId = document.getElementById('gop_sessionId').value;
        var requestDetailsStr = document.getElementById('requestDetail').value;

        //trim string
        var trimmed = requestDetailsStr.replace(/\{/g, "");
        var parts = trimmed.split("}");

        //sanitize strings
        for(i = 0; i < parts.length; i += 1) {
            parts[i] = parts[i].replace(/^\s+|\s+$/g, "");
        }

        //build get offer requests
        var getOffReqs = [];
        for(var i = 0; i < parts.length; i += 1) {
            var getofReqObj = parseGetOfferReq(parts[i]);
            if (getofReqObj) {
                getOffReqs.push(getofReqObj);

        InteractAPI.getOffersForMultipleInteractionPoints
        (ssId, getOffReqs, callback);

}
```

- **session ID** - a string identifying the current session.
- **requestDetailsStr** - a string providing an array of `GetOfferRequest` objects.

  Each `GetOfferRequest` object specifies:
  - **ipName** - The interaction point (IP) name for which the object is requesting offers
  - **numberRequested** - The number of unique offers it needs for the specified IP
  - **offerAttributes** - Requirements on the attributes of the delivered offers using an instance of `OfferAttributeRequirements`
  - **duplicationPolicy** - Duplication policy ID for the offers that will be delivered

    Duplication policies determine whether duplicated offers will be returned across different interaction points in a single method call. (*Within* an individual interaction point, duplicated offers are never returned.) Currently, two duplication policies are supported.

- NO_DUPLICATION (ID value = 1). None of the offers that have been included in the preceding `GetOfferRequest` instances will be included in this `GetOfferRequest` instance (that is, Interact will apply de-duplication).
- ALLOW_DUPLICATION (ID value = 2). Any of the offers satisfying the requirements specified in this `GetOfferRequest` instance will be included. The offers that have been included in the preceding `GetOfferRequest` instances will not be reconciled.

– **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

The order of requests in the array parameter is also the priority order when offers are being delivered.

For example, suppose the IPs in the request are IP1, then IP2, that no duplicated offers are allowed (a duplication policy ID = 1), and each is requesting two offers. If Interact finds offers A, B, and C for IP1 and offers A and D for IP2, the response will contain offers A and B for IP1, and only offer D for IP2.

Also note that when the duplication policy ID is 1, the offers that have been delivered via an IP with higher priority will not be delivered via this IP.

The `getOffersForMultipleInteractionPoints` method waits the number of milliseconds defined in the `segmentationMaxWaitTimeInMS` property for all re-segmentation to complete before running. Therefore, if you call a `postEvent` method which triggers a re-segmentation or a `setAudience` method immediately before a `getOffers` call, there may be a delay.

Return value

The runtime server responds to `getOffersForMultipleInteractionPoints` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- Array of OfferList
- Profile
- SessionID
- StatusCode

setAudience

The `setAudience` method enables you to set the audience ID and level for a visitor.

```
function callSetAudience(commandsToExecute, callback) {

        var ssId = document.getElementById('sa_sessionId').value;
        var audId = document.getElementById('sa_audienceId').value;
        var audLevel = document.getElementById('sa_audienceLevel').value;
        var params = document.getElementById('sa_parameters').value;

        InteractAPI.setAudience(ssId, getNameValuePairs(audId),audLevel,
                                      getNameValuePairs(params), callback);

}
```

- **sessionID** - a string identifying the session ID.
- **audienceID** - an array of `NameValuePairImpl` objects that defines the audience ID.
- **audienceLevel** - a string that defines the audience level.

- **parameters** - NameValuePairImpl objects identifying any parameters that need to be passed with setAudience. These values are stored in the session data and can be used for segmentation.

  You must have a value for every column in your profile. This is a superset of all columns in all the tables defined for the interactive channel and any real-time data. If you have already populated all the session data with startSession or postEvent, you do not need to send new parameters.
- **callback** - If the method was successful, the callback function calls onSuccess. If the method failed, the callback function calls onError.

The setAudience method triggers a re-segmentation. The getOffers method waits for re-segmentation to finish before running. Therefore, if you call a setAudience method immediately before a getOffers call, there may be a delay.

The setAudience method also loads the profile data for the audience ID. You can use the setAudience method to force a reload of the same profile data loaded by the startSession method.

The setAudience method reloads the while list and the black list table in an existing session . You can use the setAudience method with the parameters UACIPurgePriorWhiteListOnLoad and UACIPurgePriorBlackListOnLoad to reload the white list table and black list table in an existing session.

By default, when the setAudience method is called, all the contents of the black list is removed. You can set the UACIPurgePriorWhiteListOnLoad and UACIPurgePriorBlackListOnLoad parameters in the setAudience call as follows:

- If you set UACIPurgePriorBlackListOnLoad= 0, all the contents of the white list table are preserved.
- If you set UACIPurgePriorWhiteListOnLoad= 1 the contents of the table are removed and the contents of the white list or black list for the audience ID will be loaded from the database. Once completed, re-segmentation will start.

Return value

The runtime server responds to setAudience with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList
- Profile
- SessionID
- StatusCode

getProfile

The getProfile method enables you to retrieve the profile and temporal information about the visitor visiting the touchpoint.

```
function callGetProfile(commandsToExecute, callback) {

        var ssId = document.getElementById('gp_sessionId').value;

        InteractAPI.getProfile(ssId, callback);

}
```

- **session ID**-a string identifying the session ID.

- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

Return value

The runtime server responds to `getProfile` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList
- ProfileRecord
- SessionID
- StatusCode

endSession

The `endSession` method marks the end of the runtime session. When the runtime server receives this method, the runtime server logs to history, clears memory, and so on.

```
function callEndSession(commandsToExecute, callback) {

        var ssId = document.getElementById('es_sessionId').value;

        InteractAPI.endSession(ssId, callback);

}
```

- **session ID** - Unique string identifying the session.
- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

If the endSession method is not called, runtime sessions timeout. The timeout period is configurable with the `sessionTimeout` property.

Return value

The runtime server responds to the `endSession` method with the `Response` object with the following attributes populated:

- SessionID
- ApiVersion
- OfferList
- Profile
- StatusCode
- AdvisoryMessages

setDebug

The `setDebug` method enables you to set the logging verbosity level for all code paths for the session.

```
function callSetDebug(commandsToExecute, callback) {

        var ssId = document.getElementById('sd_sessionId').value;
        var isDebug = document.getElementById('isDebug').value;
```

```
        InteractAPI.setDebug(ssId, isDebug, callback);

}
```

- **sessionID**-a string which identifies the session ID.
- **debug**-a boolean which enables or disables debug information. Valid values are `true` or `false`. If true, Interact logs debug information to the runtime server log.
- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

Return value

The runtime server responds to `setDebug` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList
- Profile
- SessionID
- StatusCode

getVersion

The `getVersion` method returns the version of the current implementation of the Interact runtime server.

```
function callGetVersion(commandsToExecute, callback) {

        InteractAPI.getVersion(callback);


}
```

Best practice is to use this method when you initialize the touchpoint with the Interact API.

- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

Return value

The runtime server responds to the `getVersion` with a Response object with the following attributes populated:

- AdvisoryMessages
- ApiVersion
- OfferList
- Profile
- SessionID
- StatusCode

executeBatch

The `executeBatch` method enables you to execute several methods with a single request to the runtime server.

```
function callExecuteBatch(commandsToExecute, callback) {

        if (!commandsToExecute)
            return ;

        InteractAPI.executeBatch(commandsToExecute.ssid,
        commandsToExecute.commands, callback);
}
```

- **session ID**-A string identifying the session ID. This session ID is used for all commands run by this method call.
- **commands**-An array of command objects, one for each command you want to perform.
- **callback** - If the method was successful, the callback function calls `onSuccess`. If the method failed, the callback function calls `onError`.

The result of calling this method is equivalent to explicitly calling each method in the Command array. This method minimizes the number of actual requests to the runtime server. The runtime server runs each method serially; for each call, any error or warnings are captured in the Response object that corresponds to that method call. If an error is encountered, the executeBatch continues with the rest of the calls in the batch. If the running of any method results in an error, the top level status for the BatchResponse object reflects that error. If no error occurred, the top level status reflects any warnings that may have occurred. If no warning occurred, then the top level status reflects a successful run of the batch.

Return value

The runtime server responds to the `executeBatch` with a BatchResponse object.

JavaScript API example

```
function isJavaScriptAPISelected() {
    var radios = document.getElementsByName('api');
    for (var i = 0, length = radios.length; i < length; i++) {
        if (radios[i].checked) {
            if (radios[i].value === 'JavaScript')
                return true ;
            else  // only one radio can be logically checked
                break;
        }
    }
    return false;
}

function processFormForJSInvocation(e) {

    if (!isJavaScriptAPISelected())
        return;

    if (e.preventDefault) e.preventDefault();

    var serverurl = document.getElementById('serviceUrl').value ;
    InteractAPI.init( { "url" : serverurl } );

    var commandsToExecute = { "ssid" : null, "commands" : [] };
    var callback = InteractAPI.Callback.create(onSuccess, onError);

    callStartSession(commandsToExecute, callback);
    callGetOffers(commandsToExecute, callback);
    callGetOffersForMultipleInteractionPoints(commandsToExecute, callback);
    callPostEvent(commandsToExecute, callback);
    callSetAudience(commandsToExecute, callback);
```

```
    callGetProfile(commandsToExecute, callback);
    callEndSession(commandsToExecute, callback);
    callSetDebug(commandsToExecute, callback);
    callGetVersion(commandsToExecute, callback);

    callExecuteBatch(commandsToExecute, callback);

    // You must return false to prevent the default form behavior
    return false;
}

function callStartSession(commandsToExecute, callback) {

    //read configured start session
    var ssId = document.getElementById('ss_sessionId').value;
    var icName = document.getElementById('ic').value;
    var audId = document.getElementById('audienceId').value;
    var audLevel = document.getElementById('audienceLevel').value;
    var params = document.getElementById('ss_parameters').value;
    var relyOldSs = document.getElementById('relyOnOldSession').value;
    var debug = document.getElementById('ss_isDebug').value;

    if (commandsToExecute && !commandsToExecute.ssid) {
        commandsToExecute.ssid = ssId;
    }

    if (commandsToExecute && commandsToExecute.commands) {
        commandsToExecute.commands.push(InteractAPI.CommandUtil.
                                createStartSessionCmd(
                                icName, getNameValuePairs(audId),
                                audLevel, getNameValuePairs(params),
                                relyOldSs, debug));
    }
    else {
        InteractAPI.startSession(ssId, icName,
                               getNameValuePairs(audId), audLevel,
                               getNameValuePairs(params), relyOldSs,
                               debug, callback) ;
    }

}

function callGetOffers(commandsToExecute, callback) {

    var ssId = document.getElementById('go_sessionId').value;
    var ip = document.getElementById('go_ipoint').value;
    var nofRequested = 5 ;
    var nreqString = document.getElementById('offersRequested').value;
    if (!nreqString && nreqString!== '')
        nofRequested = Number(nreqString);

    if (commandsToExecute && !commandsToExecute.ssid) {
        commandsToExecute.ssid = ssId;
    }

    if (commandsToExecute && commandsToExecute.commands) {
        commandsToExecute.commands.push(InteractAPI.CommandUtil.
                                createGetOffersCmd(ip, nofRequested));
    }
    else {
        InteractAPI.getOffers(ssId, ip, nofRequested, callback);
    }
}

function callPostEvent(commandsToExecute, callback) {

    var ssId = document.getElementById('pe_sessionId').value;
```

```
    var ev = document.getElementById('event').value;
    var params = document.getElementById('parameters').value;

    if (commandsToExecute && !commandsToExecute.ssid) {
        commandsToExecute.ssid = ssId;
    }

    if (commandsToExecute && commandsToExecute.commands) {
        commandsToExecute.commands.push(InteractAPI.
                                    CommandUtil.createPostEventCmd
                                    (ev, getNameValuePairs(params)));
    }
    else {
        InteractAPI.postEvent(ssId, ev, getNameValuePairs(params), callback);
    }
}

function callGetOffersForMultipleInteractionPoints
(commandsToExecute, callback) {

    var ssId = document.getElementById('gop_sessionId').value;
    var requestDetailsStr = document.getElementById('requestDetail').value;

    //trim string
    var trimmed = requestDetailsStr.replace(/\{/g, "");
    var parts = trimmed.split("}");

    //sanitize strings
    for(i = 0; i < parts.length; i += 1) {
        parts[i] = parts[i].replace(/^\s+|\s+$/g, "");
    }

    //build get offer requests
    var getOffReqs = [];
    for(var i = 0; i < parts.length; i += 1) {
        var getofReqObj = parseGetOfferReq(parts[i]);
        if (getofReqObj) {
            getOffReqs.push(getofReqObj);
        }
    }

    if (commandsToExecute && !commandsToExecute.ssid) {
        commandsToExecute.ssid = ssId;
    }

    if (commandsToExecute && commandsToExecute.commands) {
        commandsToExecute.commands.push(InteractAPI.CommandUtil.
                                    createGetOffersForMultiple
                                    InteractionPointsCmd(getOffReqs));
    }
    else {
        InteractAPI.getOffersForMultipleInteractionPoints
                                    (ssId, getOffReqs, callback);
    }
}

function parseGetOfferReq(ofReqStr) {

    if (!ofReqStr || ofReqStr==="")
        return null;

    var posIp = ofReqStr.indexOf(',');
    var ip = ofReqStr.substring(0,posIp);
    var posNmReq = ofReqStr.indexOf(',', posIp+1);
    var numReq = ofReqStr.substring(posIp+1,posNmReq);
    var posDup = ofReqStr.indexOf(',', posNmReq+1);
    var dupPolicy = null;
```

```
    var reqAttributes = null;

    if (posDup===-1)
        dupPolicy = ofReqStr.substring(posNmReq+1);
    else
        dupPolicy = ofReqStr.substring(posNmReq+1,posDup);

    //check if request string has attributes
    var reqAttrPos = ofReqStr.search(/\(/g);
    if (reqAttrPos!==-1) {
        var reqAttributesStr = ofReqStr.substring(reqAttrPos);
        reqAttributesStr = trimString(reqAttributesStr);
        reqAttributesStr = removeOpenCloseBrackets(reqAttributesStr);
        reqAttributes = parseReqAttributes(reqAttributesStr);
    }

   return InteractAPI.GetOfferRequest.create(ip, parseInt(numReq),
                                parseInt(dupPolicy), reqAttributes);
}

//trim string
function trimString(strToTrim) {
    if (strToTrim)
        return strToTrim.replace(/^\s+|\s+$/g, "");
    else
        return null;
}

function trimStrArray(strArray) {
    if (!strArray) return ;
    for(var i = 0; i < strArray.length; i += 1) {
        strArray[i] = trimString(strArray[i]);
    }
}

//remove open and close brackets in the end
function removeOpenCloseBrackets(strToUpdate) {
    if (strToUpdate)
        return strToUpdate.replace(/^\(+|\)+$/g, "");
    else
        return null;
}

function parseReqAttributes(ofReqAttrStr) {

    //sanitize string
    ofReqAttrStr = trimString(ofReqAttrStr);
    ofReqAttrStr = removeOpenCloseBrackets(ofReqAttrStr);

    if (!ofReqAttrStr || ofReqAttrStr==="")
        return null;

    //get the number requested
    var pos = ofReqAttrStr.indexOf(",");
    var numRequested = ofReqAttrStr.substring(0,pos);
    ofReqAttrStr = ofReqAttrStr.substring(pos+1);

    //first part will be attribute and rest will be child attributes
    var parts = [];
    pos = ofReqAttrStr.indexOf(",");
    if (pos!==-1) {
        parts.push(ofReqAttrStr.substring(0,pos));
        parts.push(ofReqAttrStr.substring(pos+1));
    }
    else {
        parts.push(ofReqAttrStr);
    }
```

```
    for(var i = 0; i < parts.length; i += 1) {
        //sanitize string
        parts[i] = trimString(parts[i]);
        parts[i] = removeOpenCloseBrackets(parts[i]);
        parts[i] = trimString(parts[i]);
    }

    //build list of attributes
    var attributes = [];
    var idx = 0;
    if (parts[0]) {
        var attParts = parts[0].split(";");
        for (idx=0; idx<attParts.length; idx++) {
            attParts[idx] = trimString(attParts[idx]);
            attParts[idx] = removeOpenCloseBrackets(attParts[idx]);
            attParts[idx] = trimString(attParts[idx]);

            var atrObj = parseAttribute(attParts[idx]);
            if (atrObj) attributes.push(atrObj);
        }
    }

    //build list of child attributes
    var childAttributes = [];
    if (parts[1]) {
        var childAttParts = parts[1].split(")");
        for (idx=0; idx<childAttParts.length; idx++) {

            childAttParts[idx] = trimString(childAttParts[idx]);
            childAttParts[idx] = removeOpenCloseBrackets(childAttParts[idx]);
            childAttParts[idx] = trimString(childAttParts[idx]);


            //get the number requested
            var noReqPos = childAttParts[idx].indexOf(",");
            var numReqAt = childAttParts[idx].substring(0,noReqPos);
            childAttParts[idx] = childAttParts[idx].substring(noReqPos+1);
            childAttParts[idx] = trimString(childAttParts[idx]);


            var atrObjParsed = parseAttribute(childAttParts[idx]);
            if (atrObjParsed) {
                var childReq = InteractAPI.OfferAttributeRequirements.create
                (parseInt(numReqAt), [atrObjParsed], null);
                childAttributes.push(childReq);
            }
        }
    }

    return InteractAPI.OfferAttributeRequirements.create(parseInt(numRequested),
    attributes, childAttributes);
}

function parseAttribute(attStr) {

    attStr = trimString(attStr);

    if (!attStr || attStr==="")
        return null;

    var pos1 = attStr.indexOf("=");
    var pos2 = attStr.indexOf("|");
    var nvp = InteractAPI.NameValuePair.create
                          ( attStr.substring(0,pos1),
                            attStr.substring(pos1+1, pos2),
                            attStr.substring(pos2+1));
```

```
        return nvp;
    }
function callSetAudience(commandsToExecute, callback) {
        if (!document.getElementById('checkSetAudience').checked)
            return ;

        var ssId = document.getElementById('sa_sessionId').value;
        var audId = document.getElementById('sa_audienceId').value;
        var audLevel = document.getElementById('sa_audienceLevel').value;
        var params = document.getElementById('sa_parameters').value;

        if (commandsToExecute && !commandsToExecute.ssid) {
            commandsToExecute.ssid = ssId;
        }

        if (commandsToExecute && commandsToExecute.commands) {
            commandsToExecute.commands.push(InteractAPI.CommandUtil.
                                            createSetAudienceCmd
            (getNameValuePairs(audId), audLevel, getNameValuePairs(params)));
        }
        else {
            InteractAPI.setAudience(ssId, getNameValuePairs(audId),
                                    audLevel, getNameValuePairs(params),
                                    callback);
        }
    }

    function callGetProfile(commandsToExecute, callback) {

        var ssId = document.getElementById('gp_sessionId').value;

        if (commandsToExecute && !commandsToExecute.ssid) {
            commandsToExecute.ssid = ssId;
        }

        if (commandsToExecute && commandsToExecute.commands) {
            commandsToExecute.commands.push(InteractAPI.CommandUtil.
            createGetProfileCmd());
        }
        else {
            InteractAPI.getProfile(ssId, callback);
        }
    }

    function callEndSession(commandsToExecute, callback) {

        var ssId = document.getElementById('es_sessionId').value;

        if (commandsToExecute && !commandsToExecute.ssid) {
            commandsToExecute.ssid = ssId;
        }

        if (commandsToExecute && commandsToExecute.commands) {
            commandsToExecute.commands.push(InteractAPI.CommandUtil.
            createEndSessionCmd());
        }
        else {
            InteractAPI.endSession(ssId, callback);
        }
    }

    function callSetDebug(commandsToExecute, callback) {

        var ssId = document.getElementById('sd_sessionId').value;
        var isDebug = document.getElementById('isDebug').value;

        if (commandsToExecute && !commandsToExecute.ssid) {
```

```
        commandsToExecute.ssid = ssId;
    }

    if (commandsToExecute && commandsToExecute.commands) {
        commandsToExecute.commands.push(InteractAPI.CommandUtil.
        createSetDebugCmd(isDebug));
    }
    else {
        InteractAPI.setDebug(ssId, isDebug, callback);
    }
}

function callGetVersion(commandsToExecute, callback) {

    if (commandsToExecute && commandsToExecute.commands) {
        commandsToExecute.commands.push(InteractAPI.CommandUtil.
        createGetVersionCmd());
    }
    else {
        InteractAPI.getVersion(callback);
    }
}

function callExecuteBatch(commandsToExecute, callback) {

    if (!commandsToExecute)
        return ;

    InteractAPI.executeBatch(commandsToExecute.ssid,
    commandsToExecute.commands, callback);
}

function getNameValuePairs(parameters) {

    if (parameters === '')
        return null ;

    var parts = parameters.split(';');
    var nvpArray = new Array(parts.length);

    for(i = 0; i < parts.length; i += 1) {
        var nvp = parts[i].split(',') ;
        var value = null;
        if (nvp[2]===InteractAPI.NameValuePair.prototype.TypeEnum.NUMERIC) {
            if (isNaN(nvp[1])) {
                value = nvp[1];  //a non number was provided as number,
                pass it to API as it is
            }
            else {
                value = Number(nvp[1]);
            }
        }
        else {
            value = nvp[1];
        }
        //special handling NULL value
        if (value && typeof value === 'string') {
            if (value.toUpperCase() === 'NULL') {
                value = null;
            }
        }
        nvpArray[i] = InteractAPI.NameValuePair.create(nvp[0], value, nvp[2]) ;
    }

    return nvpArray;
}
```

```
function showResponse(textDisplay) {
    var newWin = open('','Response','height=300,width=300,titlebar=no,
    scrollbars=yes,toolbar=no,
   resizable=yes,menubar=no,location=no,status=no');

    if (newWin.locationbar !== 'undefined' && newWin.locationbar
    && newWin.locationbar.visible)
        newWin.locationbar.visible = false;

    var displayHTML = '<META HTTP-EQUIV="Content-Type"
    CONTENT="text/html; charset=UTF-8">
    <html><head><style>TD { border-width : thin; border-style : solid }</style.'
                        + "<script language='Javascript'>"
                        + "var desiredDomain = 'unicacorp.com'; "
                        + "if (location.href.indexOf(desiredDomain)>=0) "
                        + "{ document.domain = desiredDomain;} "
                        + "</script></head><body> "
                        + textDisplay
                        + "</body></html>" ;
    newWin.document.body.innerHTML = displayHTML;
    newWin.focus() ;
}

function onSuccess(response) {
    showResponse("********Response********<br> " + JSON.stringify(response)) ;
}

function onError(response) {
    showResponse("********Error********<br> " + response) ;
}

function formatResoponse(response) {

}

function printBatchResponse(batResponse) {

}

function printResponse(response) {

}
```

Example response JavaScript object onSuccesss

This example shows the three variables for the response JavaScript object; offerLists, messages, and profile.

`offerList` returns a non null list if you call `getOffer` or `getOffersForMultipleInteractionPoints` as an API or as part of your batch commands. You should always check null on this before you perform any operation on this variable.

You should always check the status of the `messages` JavaScript response.

`Profile` is returned non null if you use `getProfile` as an API or part of your batch commands. If you do not use `getProfile`, you can ignore this variable. You should always check null on this before you perform any operation on this variable.

```
function onSuccess(response)
InteractAPI.ResponseTransUtil._buildResponse = function(response) {
        'use strict';

        if (!response) return null;
```

```
    var offerList = null;
    //transform offerLists to JS Objects
    if (response.offerLists) {
        offerList = [];
        for (var ofListCt=0; ofListCt<response.offerLists.length;ofListCt++) {
            var ofListObj = this._buildOfferList(response.offerLists[ofListCt]);
            if (ofListObj) offerList.push(ofListObj);
        }
    }

    var messages = null;
    //transform messages to JS Objects
    if (response.messages) {
        messages = [];
        for (var msgCt=0; msgCt<response.messages.length;msgCt++) {
            var msgObj = this._buildAdvisoryMessage(response.messages[msgCt]);
            if (msgObj) messages.push(msgObj);
        }
    }

    var profile = null;
    //transform profile nvps to JS Objects
    if (response.profile) {
        profile = [];
        for (var nvpCt=0; nvpCt<response.profile.length;nvpCt++) {
            var nvpObj = this._buildNameValuePair(response.profile[nvpCt]);
            if (nvpObj) profile.push(nvpObj);
        }
    }

    return InteractAPI.Response.create(response.sessionId,
                                       response.statusCode, offerList,
                                       profile, response.version,
                                       messages) ;
};
```

Chapter 9. About the ExternalCallout API

Interact offers an extensible macro, EXTERNALCALLOUT, for use with your interactive flowcharts. This macro enables you to perform custom logic to communicate with external systems during flowchart runs. For example, if you want to calculate the credit score of a customer during a flowchart run, you can create a Java class (a callout) to do so and then use the EXTERNALCALLOUT macro in a Select process in your interactive flowchart to get the credit score from your callout.

Configuring EXTERNALCALLOUT has two major steps. First, you must create a Java class which implements the ExternalCallout API. Second, you must configure the necessary Marketing Platform configuration properties on the runtime server in the Interact | flowchart | ExternalCallouts category.

In addition to the information in this section, the JavaDoc for the ExternalCallout API is available on any Interact runtime server in the Interact/docs/externalCalloutJavaDoc directory.

IAffiniumExternalCallout interface

The ExternalCallout API is contained in the interface IAffiniumExternalCallout. You must implement the IAffiniumExternalCallout interface to use the EXTERNALCALLOUT macro.

The class that implements the IAffiniumExternalCallout should have a constructor with which it can be initialized by the runtime server.

- If there are no constructors in the class, the Java compiler creates a default constructor and this is sufficient.
- If there are constructors with arguments, a public constructor with no argument should be provided, which will be used by the runtime server.

When developing your external callout, remember the following:

- Each expression evaluation with an external callout creates a new instance of the class. You must manage thread safety issues for static members in the class.
- If your external callout uses system resources, such as files or a database connection, you must manage the connections. The runtime server does not have a facility to clean up connections automatically.

You must compile your implementation against interact_externalcallout.jar located in the lib directory of your IBM Interact runtime environment installation.

IAffiniumExternalCallout enables the runtime server to request data from your Java class. The interface consists of four methods:

- getNumberOfArguments
- getValue
- initialize
- shutdown

Adding a web service for use with the EXTERNALCALLOUT macro

Use this procedure to add a web service to use with the EXTERNALCALLOUT macro. The EXTERNALCALLOUT macro recognizes callouts only if you defined the appropriate configuration properties.

In Marketing Platform for the runtime environment, add or define the following configuration properties in the Interact > flowchart > externalCallouts category.

Configuration property	Setting
externalCallouts category	Create a category for your external callout
class	The class names for your external callout
classpath	The classpath to your external callout class files
Parameter Data category	If your external callout requires parameters, create new parameter configuration properties for them and assign each a value

getNumberOfArguments

The getNumberOfArguments method returns the number of arguments expected by the Java class with which you are integrating.

```
getNumberOfArguments()
```

Return value

The getNumberOfArguments method returns an integer.

Example

The following example shows printing the number of arguments.

```
public int getNumberOfArguments()
{
    return 0;
}
```

getValue

The getValue method performs the core functionality of the callout and returns the results.

```
getValue(audienceID, configData, arguments)
```

The getValue method requires the following parameters:

- **audienceID** - a value which identifies the audience ID.
- **configData** - a map with key-value pairs of configuration data required by the callout.
- **arguments** - the arguments required by the callout. Each argument can be a String, Double, Date, or a List of one of these. A List argument can contain null values, however, a List cannot contain, for example, a String and a Double.

  Argument type checking should be done within your implementation.

If the getValue method fails for any reason, it returns CalloutException.

Return value

The getValue method returns a list of Strings.

Example

```
public List<String> getValue(AudienceId audienceId, Map<String,
  String> configurationData, Object... arguments) throws CalloutException
{
 Long customerId = (Long) audienceId.getComponentValue("Customer");
 // now query scoreQueryUtility for the credit score of customerId
 Double score = scoreQueryUtility.query(customerId);
 String str = Double.toString(score);
 List<String> list = new LinkedList<String>();
 list.add(str);
 return list;
}
```

initialize

The `initialize` method is called once when the runtime server starts. If there are any operations which may impede performance during runtime, such as loading a database table, they should be performed by this method.

```
initialize(configData)
```

The `initialize` method requires the following parameter:

- **configData** - a map with key-value pairs of configuration data required by the callout.

  Interact reads these values from the External Callout parameters defined in the `Interact > Flowchart > External Callouts > [External Callout] > Parameter Data` category.

If the `initialize` method fails for any reason, it returns `CalloutException`.

Return value

None.

Example

```
public void initialize(Map<String, String> configurationData) throws CalloutException
{
    // configurationData has the key-value pairs specific to the environment
    // the server is running in
    // initialize scoreQueryUtility here
}
```

shutdown

The `shutdown` method is called once when the runtime server shuts down. If there are any clean up tasks required by your call out, they should run at this time.

```
shutdown(configData)
```

The `shutdown` method requires the following parameter:

- **configData**-a map with key-value pairs of configuration data required by the callout.

If the `shutdown` method fails for any reason, it returns `CalloutException`.

Return value

None.

Example

```
public void shutdown(Map<String, String> configurationData) throws CalloutException
{
    // shutdown scoreQueryUtility here
}
```

ExternalCallout API example

This example creates an external callout that gets a credit score.

Create an external callout that gets a credit score:

1. Create a file that is called `GetCreditScore.java` with the following contents. This file assumes that there is a class that is called `ScoreQueryUtility` that fetches a score from a modeling application.

```
import java.util.Map;
import com.unicacorp.interact.session.AudienceId;
import com.unicacorp.interact.flowchart.macrolang.storedobjs.IAffiniumExternalCallout;
import com.unicacorp.interact.flowchart.macrolang.storedobjs.CalloutException;
import java.util.Random;

public class GetCreditScore implements IAffiniumExternalCallout
{
 // the class that has the logic to query an external system for a customer's credit score
 private static ScoreQueryUtility scoreQueryUtility;
 public void initialize(Map<String, String> configurationData) throws CalloutException
 {
  // configurationData has the key- value pairs specific to the environment the server is running in
  // initialize scoreQueryUtility here
 }

 public void shutdown(Map<String, String> configurationData) throws CalloutException
 {
 // shutdown scoreQueryUtility here
 }

 public int getNumberOfArguments()
 {
  // do not expect any additional arguments other than the customer's id
  return 0;
 }

 public List<String> getValue(AudienceId audienceId, Map<String, String> configurationData,
  Object... arguments) throws CalloutException
 {
  Long customerId = (Long) audienceId.getComponentValue("Customer");
  // now query scoreQueryUtility for the credit score of customerId
  Double score = scoreQueryUtility.query(customerId);
  String str = Double.toString(score);
  List<String> list = new LinkedList<String>();
  list.add(str);
  return list;
 }
}
```

2. Compile `GetCreditScore.java` to `GetCreditScore.class`.
3. Create a JAR file called `creditscore.jar` containing `GetCreditScore.class` and the other class files it uses.

4. Copy the JAR file to some location on the runtime server, for example `/data/interact/creditscore.jar`.
5. Create an External Callout with name `GetCreditScore` and classpath as `/data/interact/creditscore.jar` in the `externalCallouts` category on the Manage Configurations page.
6. In an interactive flowchart, the callout can be used as `EXTERNALCALLOUT('GetCreditScore')`.

IInteractProfileDataService interface

The Profile Data Services API is contained in the interface `iInteractProfileDataService`. This interface allows you to import hierarchical data into an Interact session via one or more external data sources (such as a flat file, web service, and so on) at the time the Interact session starts or the audience ID of an Interact session changes.

To develop hierarchical data import using the Profile Data Services API, you must write a Java class that pulls information from any data source and maps it to an `ISessionDataRootNode` object, then refer to that mapped data using the EXTERNALCALLOUT macro in a Select process of an interactive flowchart.

You must compile your implementation against `interact_externalcallout.jar` located in the `lib` directory of your IBM Interact runtime environment installation.

For a complete set of Javadoc documentation for using this interface, view the files in *Interact_home*`/docs/externalCalloutJavaDoc` with any web browser.

For a sample implementation of how to use the Profile Data Service, including commented descriptions of how the example was implemented, see *Interact_home*`/samples/externalcallout/XMLProfileDataService.java`.

Note: The sample implementation is intended to be used only as an example. You should not use this sample in your implementation.

Adding a data source for use with Profile Data Services

Use this procedure to add a data source to use with the Profile Data Services.

The `EXTERNALCALLOUT` macro recognizes a data source for Profile Data Services hierarchical data import only if you defined the appropriate configuration properties.

In Marketing Platform for the runtime environment, add or define the following configuration properties in the `Interact > profile > Audience Levels > [AudienceLevelName] > Profile Data Services` category.

Configuration property	Setting
`New category Name` category	The name of the data source you are defining. The name that you enter here must be unique among the data sources for the same audience level.
`enabled`	Indicates whether the data source is enabled for the audience level in which it is defined.
`className`	The fully qualified name of the data source class that implements `IInteractProfileDataService`

Configuration property	Setting
classPath	The classpath to your Profile Data Services class files. If you omit it, the class path of the containing application server is used by default.
priority category	The priority of this data source within this audience level. It must be a unique value among all of the data sources for each audience level. (That is, if a priority is set to 100 for a data source, no other data source within the audience level can have a priority of 100.)

IParameterizableCallout interface

The Parameterizable Callout API is contained in the interface IParameterizableCallout.

This interface is the base interface of the exposed API interfaces that can accept parameters from the configuration via Marketing Platform. Since this is a base interface, it should not be directly implemented. The parameter are retrieved from the child nodes of the Parameter Data node under the category that references this implementation. In the following example, ESB is a custom implementation of the profile data service, which in turn implements the IParameterizableCallout interface. The parameters endPoint and login, together with their values are passed into this implementation class when Interact engine tries to initialize and terminate it.

```
Profile Data Services
...ESB
   ...Parameter Data
      ...endPoint
      ...login
```

The interface consists of two methods:

- initialize
- shutdown

initialize

The initialize method initializes this implementation class.

```
void initialize(java.util.Map<java.lang.String,java.lang.String> configurationData)
               throws CalloutException
```

The initialize method requires the following parameter:

- **configurationData** - a map with name value pairs of parameters configured by users

Throws

CalloutException

shutdown

The shutdown method shuts down this implementation class.

```
void shutdown(java.util.Map<java.lang.String,java.lang.String> configurationData)
             throws CalloutException
```

The shutdown method requires the following parameter:

- **configurationData** - a map with name value pairs of parameters configured by users

Throws

`CalloutException`

ITriggeredMessageAction interface

The Triggered Message Action API is contained in the interface `ITriggeredMessageAction`. This interface allows you to get and set the name of this instance.

The `ITriggeredMessageAction` interface serves as a base interface for other interfaces and should never be directly implemented.

The interface consists of two methods:

- `getName`
- `setName`

getName

The `getName` method returns the name of the `ITriggeredMessageAction` instance.

```
java.lang.String getName()
```

setName

The `setName` method sets the name of the `ITriggeredMessageAction` instance.

```
void setName(java.lang.String name)
```

While you initialize the implementation class of this interface, Interact sets the name of the interface with the name given in the configuration UI.

In the following example, the name of this gateway is `InteractLog`.

```
triggeredMessage
           ...gateways
              ...InteractLog
```

The `setName` method requires the following parameter:

- name - the name you want to set for the `ITriggeredMessageAction` instance.

IChannelSelector interface

The Channel Selector API is contained in the interface `IChannelSelector`. This interface allows you to selects the outbound channels based on the offer to be sent and session attributes.

For a sample implementation of how to use the Triggered Message Action, including commented descriptions of how the example was implemented, see *Interact_home*`/samples/triggeredmessage/SampleChannelSelector.java`.

Note: The sample implementation is intended to be used only as an example. You should not use this sample in your implementation.

You should try to use this implementation instead of writing your own.

The interface consists of one method:

- selectChannels

selectChannels

The selectChannels method selects the outbound channels that the passed-in offer should be sent to with the IChannelSelector interface.

```
java.util.List<java.lang.String> selectChannels
                   (java.util.Map<java.lang.String,java.util.Map<java.lang.String,
                                  java.lang.Object>> availableChannels,
                                  com.unicacorp.interact.api.Offer offer,
                                  com.unicacorp.interact.treatment.
                                  optimization.IInteractSessionData sessionData)
```

Interact tries to send this offer to all those returned channels.

The selectChannels method requires the following parameters:

- **availableChannels** - a map of available outbound channels, which are configured in the Triggered Message UI in the Interact design time settings. In each entry of the map, the key is the name of the channel and the value is the configured parameters for that channel in the Interact design time. The iteration order of this map matches the order defined on that UI. If Profile Preferred Channel is used on the Triggered Message UI, it is replaced by the actual channel before this method is invoked. In addition, if the same channel occurs multiple times on the UI, only the occurrence with the highest priority is kept and all the duplicates are removed.
- **offer** - the offer to be delivered
- **sessionData** - the attributes currently stored in the associated Interact session

IDispatcher interface

The Dispatcher API is contained in the interface IDispatcher. This interface sends offers to targeted gateways.

Since there is only one instance of this class for each configured dispatcher, the implementation of this interface must be stateless from the perspective of Interact.

For a sample implementation of how to use the Triggered Message Action, including commented descriptions of how the example was implemented, see *Interact_home*/samples/triggeredmessage/SampleDispatcher.java.

Note: The sample implementation is intended to be used only as an example. You should not use this sample in your implementation.

You should try to use this implementation instead of writing your own.

The interface consists of one method:

- dispatch

dispatch

The dispatch method sends offers to the target gateways in the IDispatcher interface.

```
boolean dispatch(java.lang.String channel,
            java.lang.String gatewayName,
            java.util.Collection<com.unicacorp.interact.api.Offer> offers,
            com.unicacorp.interact.api.NameValuePair[] profileData)
              throws com.unicacorp.interact.exceptions.InteractException
```

Once outbound channels are selected for a candidate offer, Interact tries to send the candidate offers to the handlers associated to the channel. The handlers are attempted based on their defined priorities from high to low. For each handler, Interact invokes this method of the configured dispatcher. It is up to the implementation of this dispatcher instance how to route the offer to the target gateway, which is configured in the same handler. If there are multiple offers sent to the same handler as a result of the same triggered message evaluation, Interact tries to send all these offers in one batch.

The `dispatch` method requires the following parameters:

- **channel** - the outbound channel these offers are sent to
- **gatewayName** - the name of the target gateway
- **offers** - the offers to be sent to the gateway in a batch
- **profileData** - profile attributes populated by `IGateway.validate` and are passed to `IGateway.deliver`

Return value

The `dispatch` method returns if the dispatch succeeded or failed

Throws

`com.unicacorp.interact.exceptions.InteractException`

IGateway interface

The Gateway API is contained in the interface `IGateway`. This interface receives offers from Interact and sends the offers to their destination.

Each implementation of this interface communicates with a particular destination. The destination must perform the necessary data transformation, attribute population, and similar destination related work.

For a sample implementation of how to use the Triggered Message Action, including commented descriptions of how the example was implemented, see *Interact_home*`/samples/triggeredmessage/SampleOutboundGateway.java`.

Note: The sample implementation is intended to be used only as an example. You should not use this sample in your implementation.

The interface consists of two methods:

- `deliver`
- `validate`

deliver

The `deliver` method is called to send the offer or offers to a destination in the `IGateway` interface.

```
void deliver(java.util.Collection<com.unicacorp.interact.api.Offer> offers,
                    com.unicacorp.interact.api.NameValuePair[] profileData,
                    java.lang.String channel)
```

The `deliver` method requires the following parameters:

- **offers** - the offer to be sent

- **profileData** - the profile attributes the validate method populates in `parameterMap`
- **channel** - the outbound channel these offers will be sent to

validate

The `validate` method validates candidate offers in the `IGateway` interface.

```
java.util.Collection<com.unicacorp.interact.api.Offer> validate
    (com.unicacorp.interact.treatment.optimization.
     IInteractSessionData sessionData,
           java.util.Collection<com.unicacorp.interact.api.Offer> candidateOffers,
           java.util.Map<java.lang.String,java.lang.Object> parameterMap,
           java.lang.String channel)
```

The Interact engine invokes this method to validate the candidate offers. The implementation of this method should check the offers, offer attributes, and session attributes against the requirements of the destination to determine which offer or offers can be sent through this gateway. In addition, it may add necessary parameters into the passed-in map, which is passed back to deliver method.

The `validate` method requires the following parameters:

- **sessionData** - the attributes currently stored in the associated Interact session
- **candidateOffers** - the offers Interact selected based on the offer selection method, its parameters, and other factors. These offers are eligible to be delivered from the perspective of Interact, but still subject to the gateway.
- **parameterMap** - a map the implementation of this method should use to pass parameters to its deliver method
- **channel** - the outbound channel these offers will be sent to

Chapter 10. IBM Interact utilities

This section describes the administrative utilities available with Interact.

Run Deployment Utility (runDeployment.sh/.bat)

The runDeployment command-line tool lets you deploy an interactive channel for a specific server group from the command line, using the settings provided by a deployment.properties file that outlines all the possible parameters and is available in the same location as the runDeployment tool itself. The ability to run an interactive channel deployment from the command line is specifically useful when you are using the OffersBySQL feature. For example, you might configure a Campaign batch flowchart to run on a periodic basis. When the flowchart run completes, a trigger can be called to initialize deployment of the offers in the OffersBySQL table using this command line tool.

Description

You can find the runDeployment command-line tool installed automatically on the Interact Design Time server, in the following location:

Interact_home/interactDT/tools/deployment/runDeployment.sh (or runDeployment.bat on a Windows server)

The only argument passed in to the command is the location of a file called deployment.properties that describes all of the possible parameters needed to deploy the interactive channel/runtime server group combination. A sample file is provided for reference.

Note: Before using the runDeployment utility, you must first edit it with any text editor to provide the location of the Java runtime environment on the server. For example, you might specify *Interact_home*/jre or *Platform_home*/jre as the path, if either of those directories contains the Java runtime you want the utility to use. Alternatively, you could provide the path to any Java runtime environment that is supported for use with this release of the IBM products.

Using the runDeployment utility in a secure (SSL) environment

To use the runDeployment utility when security has been enabled on the Interact server (and therefore connecting over an SSL port), you need to add the trust store Java property as follows:

1. When you are editing the deployment.properties file for your interactive channel deployment, modify the deploymentURL property to use the secure SSL URL, as in this example:

   deploymentURL=https://<*HOST*>.<*DOMAIN*>:<*PORT*>/Campaign/interact/InvokeDeploymentServlet

2. Edit the runDeployment.sh or runDeployment.bat script using any text editor to add the following argument to the line beginning with ${JAVA_HOME}:

   -Djavax.net.ssl.trustStore=<*TrustStorePath*>

   For example, the line might look like this after you add the trust store argument:

```
${JAVA_HOME}/bin/java -Djavax.net.ssl.trustStore=<TrustStorePath>
-cp ${CLASSPATH}com.unicacorp.Campaign.interact.deployment.tools.
InvokeDeploymentClient $1
```

Replace *<TrustStorePath>* with the path to the actual SSL trust store.

Running the utility

After you have edited the utility to provide the Java runtime environment, and you have customized a copy of the `deployment.properties` file to match your environment, you can run the utility with this command:

Interact_home`/interactDT/tools/deployment/runDeployment.sh`
deployment.properties

Replace *Interact_home* with the actual value of the Interact design time installation, and replace *deployment.properties* with the actual path and name of the properties file you have customized for this deployment.

Sample `deployment.properties` file

The sample `deployment.properties` file contains a commented listing of all of the parameters you must customize to match your own environment. The sample file also contains comments that explain what each parameter is, and why you might need to customize a particular value.

```
##############################################################################
#
# The following properties feed into the InvokeDeploymentClient program.
# The program will look for a deploymentURL setting.  The program will post a
# request against that url; all other settings are posted as parameters in
# that request. The program then checks the status of the deployment and
# returns back when the deployment is at a terminal state (or if the
# specified waitTime has been reached).
#
# the output of the program will be of this format:
# <STATE> : <Misc Detail>
#
# where state can be one of the following:
# ERROR
# RUNNING
# SUCCESS
#
# Misc Detail is data that would normally populate the status message area
#  in the deployment gui of the IC summary page. NOTE: HTML tags may exist
# in the Misc Detail
#
##############################################################################

##############################################################################
# deploymentURL: url to the InvokeDeployment servlet that resides in Interact
# Design time.  should be in the following format:
# http://dt_host:port/Campaign/interact/InvokeDeploymentServlet
##############################################################################
deploymentURL=http://localhost:7001/Campaign/interact/InvokeDeploymentServlet

##############################################################################
# dtLogin:  this is the login that you would use to login to the Design Time if
# you had wanted to deploy the IC via the deployment gui inside the IC summary
# page.
##############################################################################
dtLogin=asm_admin

##############################################################################
```

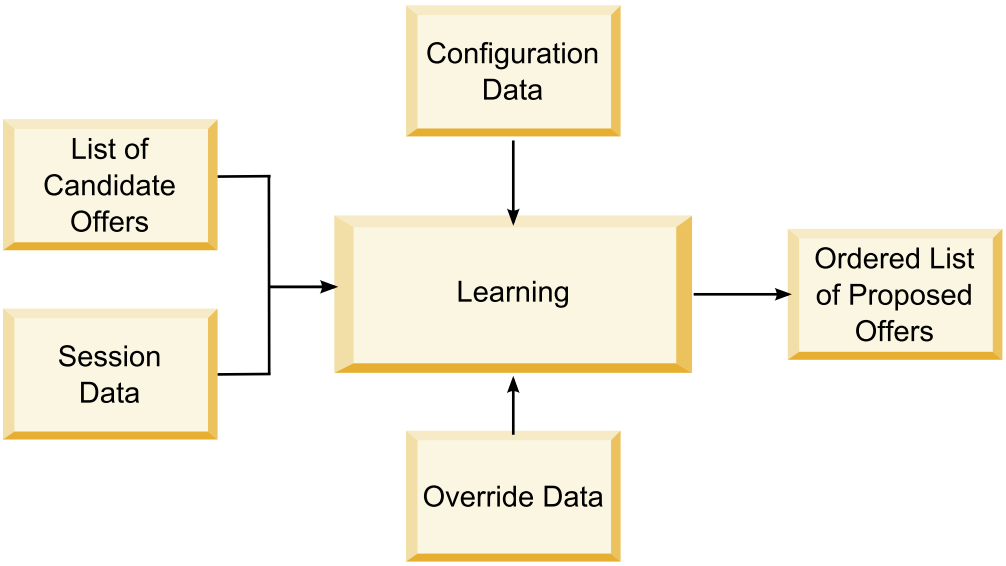
```
# dtPW:  this is the PW that goes along with the dtLogin
################################################################################
dtPW=

################################################################################
# icName:  this is the name of the Interactive Channel that you want to deploy
################################################################################
icName=ic1

################################################################################
# partition:  this is the name of the partition
################################################################################
partition=partition1

################################################################################
# request:  this is the type of request that you want this tool to execute
# currently, there two behaviors.  If the value is "deploy", then the deployment
# will be executed.  All other values would cause the tool to simply return the
# status of the last deployment of the specified IC.
################################################################################
request=deploy

################################################################################
# serverGroup:  this is the name of the server group that you would like to
# deploy the IC.
################################################################################
serverGroup=defaultServerGroup

################################################################################
# serverGroupType:  this will indicate whether or not this deployment is going
# against production server group or a test server group. 1 denotes production
# 2 denotes test.
################################################################################
serverGroupType=1

################################################################################
# rtLogin:  this is the account used to authenticate against the server group
# that you are deploying to.
################################################################################
rtLogin=asm_admin

################################################################################
# rtPW:  this is the password associated to the rtLogin
################################################################################
rtPW=

################################################################################
# waitTime:  Once the tool submits the deployment request, the tool will check
# the status of the deployment.  If the deployment has not completed (or
# failed), then the tool will continue to poll the system for the status until
# a completed state has been reached, OR until the specified waitTime (in
# seconds) has been reached.
################################################################################
waitTime=5

################################################################################
# pollTime: If the status of a deployment is still in running state, then the
# tool will continue to check the status.  It will sleep in between status
# checks a number of seconds based on the pollTime setting .
################################################################################
pollTime=3

################################################################################
# global: Setting to false will make the tool NOT deploy the global settings.
# Non-availability of the property will still deploy the global settings.
################################################################################
global=true
```

Chapter 11. About the Learning API

Interact offers a learning module which uses a naive-bayesian algorithm to monitor visitor actions and propose optimal offers (in terms of acceptance). You can implement the same Java interface with your own algorithms using the Learning API to create your own learning module.

Note: If you use External learning, the example reports regarding learning (Interactive Offer Learning Details and the Interactive Segment Lift Analysis reports) do not return valid data.

At the simplest level, the learning API provides methods to collect data from the runtime environment and to return an ordered list of recommended offers.



You can collect the following data from Interact

- Offer contact data
- Offer acceptance data
- All session data
- Campaign specific offer data
- Configuration properties defined in the `learning` category for the design environment and the `offerserving` category for the runtime environment

You can use this data in your algorithms to create a list of proposed offers. You then return a list of recommended offers, in order of highest to lowest recommendation.

Although not shown in the diagram, you can also use the learning API to collect data for your learning implementation. You can keep this data in memory, or log it to a file or database for further analysis.

After creating your Java classes, you can convert them to jar files. Once you create your jar files, you must also configure the runtime environment to recognize your

external learning module by editing configuration properties. You must copy your Java classes or jar files to every runtime server using your external learning module.

Besides the information in this guide, the JavaDoc for the learning optimizer API is available on any runtime server in the `Interact/docs/learningOptimizerJavaDoc` directory.

You must compile your implementation against `interact_learning.jar` located in the `lib` directory of your Interact runtime environment installation.

When writing your custom learning implementation, you should keep the following guidelines in mind.

- Performance is critical.
- Must work with multi-threading and be thread safe.
- Must manage all external resources with failure modes and performance in mind.
- Use exceptions, logging (log4j), and memory appropriately.

Configuring the runtime environment to recognize external learning modules

You can use the Learning Java API to write your own learning module. You must configure the runtime environment to recognize your learning utility in Marketing Platform.

You must restart the Interact runtime server for these changes to take effect.

1. In Marketing Platform for the runtime environment, edit the following configuration properties in the `Interact > offerserving` category. The configuration properties for the learning optimizer API exist in `Interact > offerserving > External Learning Config` category.

Configuration property	Setting
`optimizationType`	**ExternalLearning**
`externalLearningClass`	class name for the external learning
`externalLearningClassPath`	The path to the class or JAR files on the runtime server for the external learning. If you are using a server group and all the runtime servers reference the same instance of Marketing Platform, every server must have a copy of the class or JAR files in the same location.

2. Restart the Interact runtime server for these changes to take effect.

ILearning interface

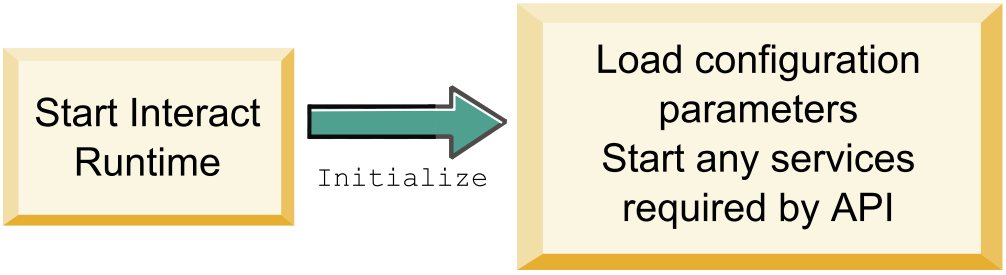
The learning API is built around the `ILearning` interface. You must implement the `ILearning` interface to support the customized logic of your learning module.

Among other things, the `ILearning` interface enables you to collect data from the runtime environment for your Java class, and to send a list of recommended offers back to the runtime server.

initialize

The `initialize` method is called once when the runtime server starts. If there are any operations that do not need to be repeated, but may impede performance during runtime, such as loading static data from a database table, they should be performed by this method.

```
initialize(ILearningConfig config, boolean debug)
```



- **config** - an `ILearningConfig` object defines all the configuration properties relevant to learning.
- **debug** - a boolean. If `true`, indicates the logging level verbosity for the runtime environment system is set to debug. For best results, select this value before writing to a log.

If the `initialize` method fails for any reason, it throws an `LearningException`.

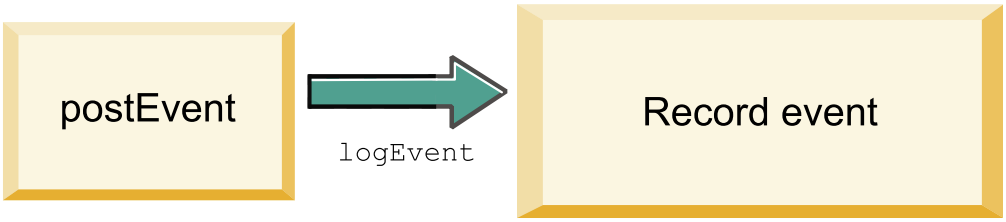
Return value

None.

logEvent

The `logEvent` method is called by the runtime server whenever the Interact API posts an event that is configured to log as a contact or response. Use this method to log contact and response data to a database or file for reporting and learning purposes. For example, if you want to algorithmically determine the likelihood of a customer accepting an offer based on criteria, use this method to log the data.

```
logEvent(ILearningContext context,
             IOffer offer,
             IClientArgs clientArgs,
             IInteractSession session,
             boolean debug)
```



- **context**-an `ILearningContext` object defining the learning context of the event, for example, contact, accept, or reject.
- **offer**-an `IOffer` object defining the offer about which this event is being logged.
- **clientArgs**-an `IClientArgs` object defining any parameters. Currently, `logEvent`, does not require any `clientArgs`, so this parameter may be empty.

- **session**-an `IInteractSession` object defining all session data.
- **debug**-a boolean. If `true`, indicates the logging level verbosity for the runtime environment system is set to debug. For best results, select this value before writing to a log.

If the `logEvent` method fails, it throws a `LearningException`.

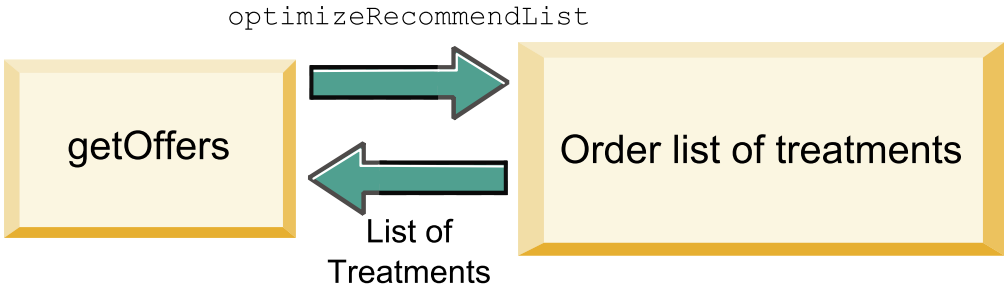
Return value

None.

optimizeRecommendList

The `optimizeRecommendList` method should take the list of recommended offers and the session data and return a list containing the requested number of offers. The `optimizeRecommendList` method should order the offers in some way, with your own learning algorithm. The list of offers must be ordered so that the offers you want to serve first are at the beginning of the list. For example, if your learning algorithm gives a low score to the best offers, the offers should be ordered 1, 2, 3. If your learning algorithm gives a high score to the best offers, the offers should be ordered 100, 99, 98.

```
optimizeRecommendList(list(ITreatment) recList,
  IClientArgs clientArg, IInteractSession session,
  boolean debug)
```



The `optimizeRecommendList` method requires the following parameters:

- **recList**-a list of the treatment objects (offers) recommended by the runtime environment.
- **clientArg**-an IClientArgs object containing at least the number of offers requested by the runtime environment.
- **session**-an IInteractSession object containing all the session data.
- **debug**-a boolean. If `true`, indicates the logging level verbosity for the runtime environment system is set to debug. For best results, select this value before writing to a log.

If the `optimizeRecommendList` method fails, it throws a `LearningException`.

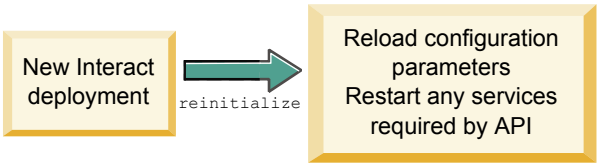
Return value

The `optimizeRecommendList` method returns a List of `ITreatment` objects.

reinitialize

The runtime environment calls the `reinitialize` method every time there is a new deployment. This method passes all learning configuration data. If you have any services required by the learning API that read configuration properties, this interface should restart them.

```
reinitialize(ILearningConfig config,
                  boolean debug)
```



- **config**-an `ILearningConfig` object which contains all the configuration properties.
- **debug**-a boolean. If `true`, indicates the logging level verbosity for the runtime environment system is set to debug. For best results, select this value before writing to a log.

If the `logEvent` method fails, it throws a `LearningException`.

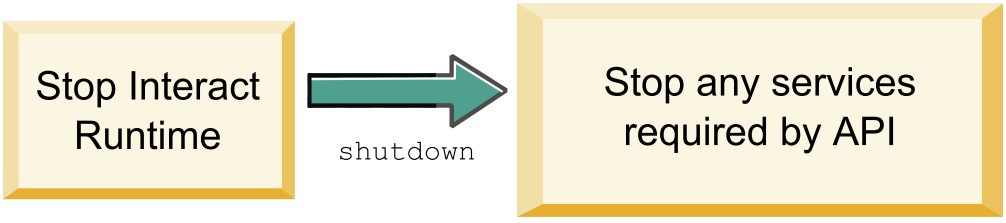
Return value

None.

shutdown

The runtime environment calls the `shutdown` method when the runtime server shuts down. If there are any clean up tasks required by your learning module, they should execute at this time.

```
shutdown(ILearningConfig config, boolean debug)
```



The `shutdown` method requires the following parameters.

- **config** - an `ILearningConfig` object which defines all the configuration properties.
- **debug** - a boolean. If `true`, indicates the logging level verbosity for the runtime environment system is set to debug. For best results, select this value before writing to a log.

If the `shutdown` method fails for any reason, it throws a `LearningException`.

Return value

None.

IAudienceID interface

The IAudienceID interface supports the IInteractSession interface. This is an interface to the audience ID. Since your audience ID may be made of several parts, this interface enables you to access all the elements of the audience ID as well as the audience level name.

getAudienceLevel

The getAudienceLevel method returns audience level.

```
getAudienceLevel()
```

Return value

The getAudienceLevel method returns a string that defines the audience level.

getComponentNames

The getComponentNames method gets a set of the names of the components which comprise the audience ID. For example, if your audience ID is consists of the values of customerName and accountID, getComponentNames would return a set containing the strings customerName and accountID.

```
getComponentNames()
```

Return value

A set of strings containing the names of the components of the audience ID.

getComponentValue

The getComponentValue method returns the value for the named component.

```
getComponentValue(String componentName)
```

- **componentName**-a string defining the name of the component for which you want to retrieve the value. This string is case insensitive.

Return value

The getComponentValue method returns an object that defines the value of the component.

IClientArgs

The IClientArgs interface supports the ILearning interface. This interface is an abstraction to cover any data passed into the server from the touchpoint that is not already covered by the session data. For example, the number of offers requested by the Interact API getOffers method. This data is stored in a map.

getValue

The getValue method returns the value of the requested map element.

```
getValue(int clientArgKey)
```

The following elements are required in the map.

- **1** - NUMBER_OF_OFFERS_REQUESTED. The number of offers requested by the getOffers method of the Interact API. This constant returns an integer.

Return value

The getValue method returns an object that defines value of the requested map constant.

IInteractSession

The IInteractSession interface supports the ILearning interface. This is an interface to the current session in the runtime environment.

getAudienceId

The getAudienceId method returns an AudienceID object. Use the IAudienceID interface to extract the values.

```
getAudienceId()
```

Return value

The getAudienceId method returns an AudienceID object.

getSessionData

The getSessionData method returns an unmodifiable map of session data where the name of the session variable is the key. The name of the session variable is always uppercased. Use the IInteractSessionData interface to extract values.

```
getSessionData()
```

Return value

The getSessionData method returns an IInteractSessionData object.

IInteractSessionData interface

The IInteractSessionData interface supports the ILearning interface. This is an interface to the runtime session data for the current visitor. Session data is stored as a list of name-value pairs. You can also use this interface to change the value of data in the runtime session.

getDataType

The getDataType method returns the data type for the specified parameter name.

```
getDataType(string parameterName)
```

Return value

The getDataType method returns an InteractDataType object. IntearctDataType is a Java enum represented by Unknown, String, Double, Date, or List.

getParameterNames

The getParameterNames method returns a set of all the names of the data in the current session.

```
getParameterNames()
```

Return value

The getParameterNames method returns a set of names for which values have been set. Each name in the set can be passed into getValue(String) to return a value.

getValue

The getValue method returns the object value corresponding to the specified parameterName. Object can either be a String, Double, or a Date.

```
getValue(parameterName)
```

The getValue method requires the following parameter:

- **parameterName**-a string defining the name of the session data name-value pair.

Return value

The getValue method returns an object containing the value of the parameter named.

setValue

The setValue method enables to set a value for the specified parameterName. The value can be can either be a String, Double, or a Date.

```
setValue(string parameterName, object value)
```

The setValue method requires the following parameters:

- **parameterName** - a string defining the name of the session data name-value pair.
- **value** - an object defining the value of the designated parameter.

Return value

None.

ILearningAttribute

The ILearningAttribute interface supports the ILearningConfig interface. This is an interface to the learning attributes defined in configuration properties in the learningAttributes category.

getName

The getName method returns the name of the learning attribute.

```
getName()
```

Return value

The getName method returns a string that defines the name of the learning attribute.

ILearningConfig

The ILearningConfig interface supports the ILearning interface. This is an interface to the configuration properties for learning. All of these methods return the value of the property.

The interface consists of 15 methods:

- **getAdditionalParameters** - returns a map of any additional properties defined in the External Learning Config category
- **getAggregateStatsIntervalInMinutes** - returns an int
- **getConfidenceLevel** - returns an int
- **getDataSourceName** - returns a string
- **getDataSourceType** - returns a string
- **getInsertRawStatsIntervalInMinutes** - returns an int
- **getLearningAttributes** - returns a list of ILearningAttribute objects
- **getMaxAttributeNames** - returns an int
- **getMaxAttributeValues** - returns an int
- **getMinPresentCountThreshold** - returns an int
- **getOtherAttributeValue** - returns a string
- **getPercentRandomSelection** - returns an int
- **getRecencyWeightingFactor** - returns a float
- **getRecencyWeightingPeriod** - returns an int
- **isPruningEnabled** - returns a boolean

ILearningContext

The ILearningContext interface supports the ILearning interface.

getLearningContext

The getLearningContext method return the constant that tells us whether or not this is a contact, accept or reject scenario.

```
getLearningContext()
```

- **1**-LOG_AS_CONTACT
- **2**-LOG_AS_ACCEPT
- **3**-LOG_AS_REJECT

4 and 5 are reserved for future use.

Return value

The getLearningContext method returns an integer.

getResponseCode

The getResponseCode method returns response code assigned to this offer. This value must exist in the UA_UsrResponseType table in the Campaign system tables.

```
getResponseCode()
```

Return value

The getResponseCode method returns a string that defines the response code.

IOffer

The IOffer interface supports the ITreatment interface. This is an interface to the offer object defined in the design environment. Use the IOffer interface to collect the offer details from the runtime environment.

getCreateDate

The getCreateDate method returns the date the offer was created.

```
getCreateDate()
```

Return value

The getCreateDate method returns a date that defines the date the offer was created.

getEffectiveDateFlag

The getEffectiveDateFlag method returns a number that defines the effective date of the offer.

```
getEffectiveDateFlag()
```

- **0**-the effective date is an absolute date, such as March 15, 2010.
- **1**-the effective date is the date of recommendation.

Return value

The getEffectiveDateFlag method returns an integer that defines the effective date of the offer.

getExpirationDateFlag

The getExpirationDateFlag method returns an integer value that describes the expiration date of the offer.

```
getExpirationDateFlag()
```

- **0**-an absolute date, for example March 15, 2010.
- **1**-some number of days after the recommendation, for example 14.
- **2**-end of month after recommendation. If an offer is presented on March 31st, the offer expires that day.

Return value

The getExpirationDateFlag method returns an integer that describes the expiration date of the offer.

getOfferAttributes

The getOfferAttributes method returns offer attributes defined for the offer as an IOfferAttributes object.

```
getOfferAttributes()
```

Return value

The getOfferAttributes method returns an IOfferAttributes object.

getOfferCode

The getOfferCode method returns the offer code of the offer as defined in Campaign.

```
getOfferCode()
```

Return value

The getOfferCode method returns an IOfferCodeobject.

getOfferDescription

The getOfferDescription method returns the description of the offer defined in Campaign.

```
getOfferDescription()
```

Return value

The getOfferDescription method returns a string.

getOfferID

The getOfferID method returns the offer ID as defined in Campaign.

```
getOfferID()
```

Return value

The getOfferID method returns a long that defines the offer ID.

getOfferName

The getOfferName method returns the name of the offer as defined in Campaign.

```
getOfferName()
```

Return value

The getOfferName method returns a string.

getUpdateDate

The getUpdateDate method returns date of when the offer was last updated.

```
getUpdateDate()
```

Return value

The getUpdateDate method returns a date that defines when the offer was last updated.

IOfferAttributes

The IOfferAttributes interface supports the IOffer interface. This is an interface to the offer attributes that are defined for an offer in the design environment. Use the IOfferAttributes interface to collect the offer attributes from the runtime environment.

getParameterNames

The getParameterNames method returns a list of the offer parameter names.

```
getParameterNames()
```

Return value

The getParameterNames method returns a set that defines the list of offer parameter names.

getValue

The getValue method returns an object that defines the value of the offer attribute.

```
getValue(String parameterName)
```

The getValue method returns value of the given offer attribute.

Return value

IOfferCode interface

The IOfferCode interface supports the ILearning interface. This is an interface to the offer code that was defined for an offer in the design environment. An offer code can be made of one to many Strings. Use the IOfferCode interface to collect the offer code from the runtime environment.

getPartCount

The getPartCount method returns the number of parts that make up an offer code.

```
getPartCount()
```

Return value

The getPartCount method returns an integer defining the number of parts of the offer code.

getParts

The getParts method gets an unmodifiable list of the offer code parts.

```
getParts()
```

Return value

The getParts method returns an unmodifiable list of the offer code parts.

LearningException

The LearningException class supports the ILearning interface. Some methods within the interface will require implementations to throw a LearningException which is a simple subclass of java.lang.Exception. It is highly recommended for debugging purposes that the LearningException be constructed with the root exception if a root exception exists.

IScoreOverride

The IScoreOverride interface supports ITreatment interface. This interface enables you to read the data defined in the score override or default offers table.

getOfferCode

The `getOfferCode` method returns the value of the offer code columns in the score override table for this audience member.

```
getOfferCode()
```

Return value

The `getOfferCode` method returns an `IOfferCode` object that defines the value of the offer code columns in the score override table.

getParameterNames

The `getParameterNames` method returns the list of parameters.

```
getParameterNames()
```

Return value

The `getParameterNames` method returns a set that defines the list of parameters.

The `IScoreOverride` method contains the following parameters. Unless otherwise stated, theses parameters are the same as the score override table.

- ADJ_EXPLORE_SCORE_COLUMN
- CELL_CODE_COLUMN
- ENABLE_STATE_ID_COLUMN
- ESTIMATED_PRESENT_COUNT - For overriding estimated present count (during offer weight calculation)
- FINAL_SCORE_COLUMN
- LIKELIHOOD_SCORE_COLUMN
- MARKETER_SCORE
- OVERRIDE_TYPE_ID_COLUMN
- PREDICATE_COLUMN - For creating a boolean expression to determine offer eligibility
- PREDICATE_SCORE - For creating an expression that results in a numeric score
- SCORE_COLUMN
- ZONE_COLUMN

You can also reference any column you add to the score override or default offers table using the same name as the column.

getValue

The `getValue` method returns the value of the zone column in the score override table for this audience member.

```
getValue(String parameterName)
```

- **parameterName**-a string defining the name of the parameter for which you want the value.

Return value

The `getValue` method returns an object defining the value of the requested parameter.

ISelectionMethod

The `ISelection` interface indicates the method used to come up with the recommended list. The default value for the Treatment object is EXTERNAL_LEARNING so you do not have to set this value. The value is ultimately stored into Detailed Contact History for reporting purposes.

You can extend this interface beyond the existing constants if you want to store the data for analysis later. For example, you could create two different learning modules and implement them on separate server groups. You could extend the `ISelection` interface to include SERVER_GROUP_1 and SERVER_GROUP_2. You could then compare the results of your two learning modules.

ITreatment interface

The `ITreatment` interface supports the `ILearning` interface as an interface to the Treatment information. A treatment represents the offer assigned to a particular cell as defined in the design environment. From this interface, you can obtain cell and offer information as well as the assigned marketing score.

getCellCode

The `getCellCode` method returns the cell code as defined in Campaign. The cell is the cell assigned to the smart segment associated with this offer.

```
getCellCode()
```

Return value

The `getCellCode` method returns a string that defines the cell code.

getCellId

The `getCellId` method returns the internal ID of the cell as defined in Campaign. The cell is the cell assigned to the smart segment associated with this offer.

```
getOfferName()
```

Return value

The `getCellId` method returns a long that defines the cell ID.

getCellName

The `getCellName` method returns the name of the cell as defined in Campaign. The cell is the cell assigned to the smart segment associated with this offer.

```
getCellName()
```

Return value

The `getCellName` method returns a string that defines the cell name.

getLearningScore

The `getLearningScore` method returns the score for this treatment.

```
getLearningScore()
```

The precedence is as follows.

1. Return the override value, if present in Override values map keyed by `IScoreoveride.PREDICATE_SCORE_COLUMN`
2. Return predicate score if the value is not null
3. Return the marketers score, if present in Override values map keyed by `IScoreoveride.SCORE`
4. Return the marketers score

Return value

The `getLearningScore` method returns an integer that defines the score determined by the learning algorithm.

getMarketerScore

The `getMarketerScore` method returns the marketer's score defined by the slider on the interaction strategy tab for the offer.

```
getMarketerScore()
```

To retrieve a marketer's score defined by the interaction strategy tab advanced options, use `getPredicateScore`.

To retrieve the marketer's score actually used by the treatment, use `getLearningScore`.

Return value

The `getMarketerScore` method returns an integer that defines the marketer's score.

getOffer

The `getOffer` method returns the offer for the treatment.

```
getOffer()
```

Return value

The `getOffer` method returns an `IOffer` object that defines the offer for this treatment.

getOverrideValues

The `getOverrideValues` method returns overrides defined in the default offers or score override table.

```
getOverrideValues()
```

Return value

The `getOverrideValues` method returns an `IScoreOverride` object.

getPredicate

The `getPredicate` method returns the predicate defined by the predicate column of the default offers table, score override table or the treatment rules advanced options.

```
getPredicate()
```

Return value

The getPredicate method returns a string that defines predicate defined by the predicate column of the default offers table, score override table or the treatment rules advanced options.

getPredicateScore

The getPredicateScore method returns the score set by the predicate column of the default offers table, score override table or the treatment rules advanced options.

```
getPredicateScore()
```

Return value

The getPredicateScore method returns a double that defines the score set by the predicate column of the default offers table, score override table, or the treatment rules advanced options.

getScore

The getScore method returns the marketing score that is defined either by the interaction strategy in Campaign or by the score override table.

```
getScore()
```

The getScore method returns one of the following:

- The marketing score of the offer as defined on the interaction strategy tab in Campaign if the enableScoreOverrideLookup property is set to false.
- The score of the offer as defined by the scoreOverrideTable if the enableScoreOverrideLookup property is set to true.

Return value

The getScore method returns an integer that represents the score of the offer.

getTreatmentCode

The getTreatmentCode method returns the treatment code.

```
getTreatmentCode()
```

Return value

The getTreatmentCode method returns a string that defines the treatment code.

setActualValueUsed

Use the setActualValueUsed method to define what values are used at various stages in the learning algorithm execution.

```
setActualValueUsed(string parmName, object value)
```

For example, if you use this method to write to the contact and response history tables, and modify the existing sample reports, you can include data from your learning algorithm in reporting.

- **parmName**-a string defining the name of the parameter you are setting.
- **value**-an object defining the value of the parameter you are setting.

Return value

None.

Learning API example

This section contains a sample implementation of the ILearningInterface. Note that this implementation is just a sample and is not designed to be used in a production environment.

This example keeps track of accept and contact counts and uses the ratio of accept to contacts for a particular offer as the acceptance probability rate for the offer. Offers not presented get higher priority for recommending. Offers with at least one contact are be ordered based on descending acceptance probability rate.

In this example, all counts are kept in memory. This is not a realistic scenario as the runtime server will run out of memory. In a real production scenario, the counts should be persisted into a database.

```
package com.unicacorp.interact.samples.learning.v2;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.unicacorp.interact.samples.learning.SampleOptimizer.MyOfferSorter;
import com.unicacorp.interact.treatment.optimization.IClientArgs;
import com.unicacorp.interact.treatment.optimization.IInteractSession;
import com.unicacorp.interact.treatment.optimization.ILearningConfig;
import com.unicacorp.interact.treatment.optimization.ILearningContext;
import com.unicacorp.interact.treatment.optimization.IOffer;
import com.unicacorp.interact.treatment.optimization.LearningException;
import com.unicacorp.interact.treatment.optimization.v2.ILearning;
import com.unicacorp.interact.treatment.optimization.v2.ITreatment;

/**
 * This is a sample implementation of the learning optimizer.
 * The interface ILearning may be found in the interact.jar library.
 *
 * To actually use this implementation, select ExternalLearning as the optimizationType in the offerServing node
 * of the Interact application within the Platform configuration.  Within the offerserving node there is also
 * an External Learning config category - within there you must set the name of the class to this:
 * com.unicacorp.interact.samples.learning.v2.SampleLearning.  Please note however, this implementation is just a sample
 * and was not designed to be used in a production environment.
 *
 *
 * This example keeps track of accept and contact counts and uses the ratio of accept to contacts
 * for a particular offer as the acceptance probability rate for the offer.
 *
 *
 * Offers not presented will get higher priority for recommending.
 * Offers with at least one contact will be ordered based on descending acceptance probability rate.
 *
 * Note: all counts are kept in memory.  This is not a realistic scenario since you would run out of memory sooner or
 * later.  In a real production scenario, the counts should be persisted into a database.
 *
 */

public class SampleLearning implements ILearning
{

    // A map of offer ids to contact count for the offer id
    private Map<Long,Integer> _offerToContactCount = new HashMap<Long, Integer>();

    // A map of offer ids to contact count for the offer id
    private Map<Long,Integer> _offerToAcceptCount = new HashMap<Long, Integer>();


    /* (non-Javadoc)
```

```
 * @see com.unicacorp.interact.treatment.optimization.v2.ILearning#initialize
 *   (com.unicacorp.interact.treatment.optimization.v2.ILearningConfig, boolean)
 */
public void initialize(ILearningConfig config, boolean debug) throws LearningException
{
    // If any remote connections are required, this is a good place to initialize those connections as this
    // method is called once at the start of the interact runtime webapp.
    // This example does not have any remote connections and prints for debugging purposes that this method will
    // be called
    System.out.println("Calling initialize for SampleLearning");
}

/* (non-Javadoc)
 * @see com.unicacorp.interact.treatment.optimization.v2.ILearning#reinitialize
 *   (com.unicacorp.interact.treatment.optimization.v2.ILearningConfig, boolean)
 */
public void reinitialize(ILearningConfig config, boolean debug) throws LearningException
{
    // If an IC is deployed, this reinitialize method is called to allow the implementation to
    // refresh any updated configuration settings
    System.out.println("Calling reinitialize for SampleLearning");
}


/* (non-Javadoc)
 * @see com.unicacorp.interact.treatment.optimization.v2.ILearning#logEvent
 *   (com.unicacorp.interact.treatment.optimization.v2.ILearningContext,
 *   com.unicacorp.interact.treatment.optimization.v2.IOffer,
 *   com.unicacorp.interact.treatment.optimization.v2.IClientArgs,
 *   com.unicacorp.interact.treatment.optimization.IInteractSession, boolean)
 */
public void logEvent(ILearningContext context, IOffer offer, IClientArgs clientArgs,
IInteractSession session, boolean debug) throws LearningException
{
    System.out.println("Calling logEvent for SampleLearning");


    if(context.getLearningContext()==ILearningContext.LOG_AS_CONTACT)
    {
        System.out.println("adding contact");

        // Keep track of all contacts in memory
        synchronized(_offerToAcceptCount)
        {
           Integer count = _offerToAcceptCount.get(offer.getOfferId());
           if(count == null)
               count = new Integer(1);
           else
               count++;
           _offerToAcceptCount.put(offer.getOfferId(), ++count);
        }

    }
    else if(context.getLearningContext()==ILearningContext.LOG_AS_ACCEPT)
    {
        System.out.println("adding accept");
        // Keep track of all accept counts in memory by adding to the map
        synchronized(_offerToAcceptCount)
        {
           Integer count = _offerToAcceptCount.get(offer.getOfferId());
           if(count == null)
               count = new Integer(1);
           else
               count++;
           _offerToAcceptCount.put(offer.getOfferId(), ++count);
        }
        }

}

/* (non-Javadoc)
 * @see com.unicacorp.interact.treatment.optimization.v2.ILearning#optimizeRecommendList
 *   (java.util.List, com.unicacorp.interact.treatment.optimization.v2.IClientArgs,
 *   com.unicacorp.interact.treatment.optimization.IInteractSession, boolean)
 */
public List<ITreatment> optimizeRecommendList(List<ITreatment> recList,
    IClientArgs clientArgs, IInteractSession session, boolean debug)
    throws LearningException
{
```

```
        System.out.println("Calling optimizeRecommendList for SampleLearning");

        // Sort the candidate treatments by calling the sorter defined in this class and return the sorted list
        Collections.sort(recList,new MyOfferSorter());

        // now just return what was asked for via "numberRequested" variable
        List<ITreatment> result = new ArrayList<ITreatment>();

        for(int x=0;x<(Integer)clientArgs.getValue(IClientArgs.NUMBER_OF_OFFERS_REQUESTED) && x<recList.size();x++)
        {
            result.add(recList.get(x));
        }
        return result;
    }

    /* (non-Javadoc)
     * @see com.unicacorp.interact.treatment.optimization.v2.ILearning#shutdown
     *   (com.unicacorp.interact.treatment.optimization.v2.ILearningConfig, boolean)
     */
    public void shutdown(ILearningConfig config, boolean debug) throws LearningException
    {
        // If any remote connections exist, this would be a good place to gracefully
        // disconnect from them as this method is called at the shutdown of the Interact runtime
        // webapp.   For this example, there is nothing really to do
        // except print out a statement for debugging.
        System.out.println("Calling shutdown for SampleLearning");

    }
    // Sort by:
    // 1. offers with zero contacts  - for ties, order is based on original input
    // 2. descending accept probability rate - for ties, order is based on original input

    public class MyOfferSorter implements Comparator<ITreatment>
    {
        private static final long serialVersionUID = 1L;

        /* (non-Javadoc)
         * @see java.lang.Comparable#compareTo(java.lang.Object)
         */
        public int compare(ITreatment treatment1, ITreatment treatment2)
        {

            // get contact count for both treatments
            Integer contactCount1 = _offerToContactCount.get(treatment1.getOffer().getOfferId());
            Integer contactCount2 = _offerToContactCount.get(treatment2.getOffer().getOfferId());

            // if treatment hasn't been contacted, then that wins
            if(contactCount1 == null || contactCount1 == 0)
                return -1;

            if(contactCount2 == null || contactCount2 == 0)
                return 1;

            // get accept counts
            Integer acceptCount1 = _offerToAcceptCount.get(treatment1.getOffer().getOfferId());
            Integer acceptCount2 = _offerToAcceptCount.get(treatment2.getOffer().getOfferId());

            float acceptProbability1 = (float) acceptCount1 / (float) contactCount1;
            float acceptProbability2 = (float) acceptCount2 / (float) contactCount2;

            // descending order
            return (int) (acceptProbability2 - acceptProbability1);

        }
    }

}
```

Chapter 12. IBM Interact WSDL

The Interact installation includes two WDSL (Web Services Description Language) XML files that describe the available web services and how to access them. You can view these files in your Interact home directory, and an example is shown here.

After you have installed Interact, you can find the Interact WSDL files in the following location:

- <*Interact_home*>/conf/InteractService.wsdl
- <*Interact_home*>/conf/InteractAdminService.wsdl

With each software release or fix pack, there can be changes to the Interact WDSL. See the *Interact Release Notes* or the readme files with the release for details.

A copy of the `InteractService.wsdl` is shown here for reference. To ensure that you are using the latest information, see the WDSL files that are installed with Interact.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:ns0="http://soap.api.interact.unicacorp.com" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" bloop="http://api.interact.unicacorp.com/xsd"
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" targetNamespace="http://soap.api.interact.unicacorp.com">
 <wsdl:types>
  <xs:schema xmlns:ns="http://soap.api.interact.unicacorp.com" attributeFormDefault="qualified"
   elementFormDefault="qualified" targetNamespace="http://soap.api.interact.unicacorp.com">
   <xs:element name="executeBatch">
    <xs:complexType>
     <xs:sequence>
      <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
      <xs:element maxOccurs="unbounded" minOccurs="1" name="commands" nillable="false" type="ns1:CommandImpl"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="executeBatchResponse">
    <xs:complexType>
     <xs:sequence>
      <xs:element minOccurs="1" name="return" nillable="false" type="ns1:BatchResponse"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="endSession">
    <xs:complexType>
     <xs:sequence>
      <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="endSessionResponse">
    <xs:complexType>
     <xs:sequence>
      <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="getOffers">
    <xs:complexType>
     <xs:sequence>
      <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
      <xs:element minOccurs="1" name="iPoint" nillable="false" type="xs:string"/>
      <xs:element minOccurs="1" name="numberRequested" type="xs:int"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="getOffersResponse">
    <xs:complexType>
```

```
  <xs:sequence>
   <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="getProfile">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="getProfileResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="getVersionResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="postEvent">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
   <xs:element minOccurs="1" name="eventName" nillable="false" type="xs:string"/>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="eventParameters"
    nillable="true" type="ns1:NameValuePairImpl"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="postEventResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="setAudience">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="audienceID" nillable="false" type="ns1:NameValuePairImpl"/>
   <xs:element minOccurs="1" name="audienceLevel" nillable="false" type="xs:string"/>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="parameters" nillable="true" type="ns1:NameValuePairImpl"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="setAudienceResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="setDebug">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
   <xs:element minOccurs="1" name="debug" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="setDebugResponse">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="startSession">
 <xs:complexType>
  <xs:sequence>
   <xs:element minOccurs="1" name="sessionID" nillable="false" type="xs:string"/>
```

```
    <xs:element minOccurs="1" name="relyOnExistingSession" type="xs:boolean"/>
    <xs:element minOccurs="1" name="debug" type="xs:boolean"/>
    <xs:element minOccurs="1" name="interactiveChannel" nillable="false" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="1" name="audienceID" nillable="false" type="ns1:NameValuePairImpl"/>
    <xs:element minOccurs="1" name="audienceLevel" nillable="false" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="1" name="parameters" nillable="true" type="ns1:NameValuePairImpl"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="startSessionResponse">
  <xs:complexType>
   <xs:sequence>
    <xs:element minOccurs="1" name="return" nillable="false" type="ns1:Response"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
<xs:schema xmlns:ax21="http://api.interact.unicacorp.com/xsd" attributeFormDefault="qualified"
 elementFormDefault="qualified" targetNamespace="http://api.interact.unicacorp.com/xsd">
 <xs:complexType name="Command">
  <xs:sequence>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="audienceID" nillable="true" type="ax21:NameValuePair"/>
   <xs:element minOccurs="1" name="audienceLevel" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="debug" type="xs:boolean"/>
   <xs:element minOccurs="1" name="event" nillable="true" type="xs:string"/>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="eventParameters" nillable="true" type="ax21:NameValuePair"/>
   <xs:element minOccurs="1" name="interactionPoint" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="interactiveChannel" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="methodIdentifier" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="numberRequested" type="xs:int"/>
   <xs:element minOccurs="1" name="relyOnExistingSession" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="NameValuePair">
  <xs:sequence>
   <xs:element minOccurs="1" name="name" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="valueAsDate" nillable="true" type="xs:dateTime"/>
   <xs:element minOccurs="1" name="valueAsNumeric" nillable="true" type="xs:double"/>
   <xs:element minOccurs="1" name="valueAsString" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="valueDataType" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="CommandImpl">
  <xs:sequence>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="audienceID" nillable="true" type="ax21:NameValuePairImpl"/>
   <xs:element minOccurs="1" name="audienceLevel" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="debug" type="xs:boolean"/>
   <xs:element minOccurs="1" name="event" nillable="true" type="xs:string"/>
   <xs:element maxOccurs="unbounded" minOccurs="1" name="eventParameters" nillable="true" type="ax21:NameValuePairImpl"/>
   <xs:element minOccurs="1" name="interactionPoint" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="interactiveChannel" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="methodIdentifier" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="numberRequested" type="xs:int"/>
   <xs:element minOccurs="1" name="relyOnExistingSession" type="xs:boolean"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="NameValuePairImpl">
  <xs:sequence>
   <xs:element minOccurs="1" name="name" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="valueAsDate" nillable="true" type="xs:dateTime"/>
   <xs:element minOccurs="1" name="valueAsNumeric" nillable="true" type="xs:double"/>
   <xs:element minOccurs="1" name="valueAsString" nillable="true" type="xs:string"/>
   <xs:element minOccurs="1" name="valueDataType" nillable="true" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="BatchResponse">
  <xs:sequence>
   <xs:element minOccurs="0" name="batchStatusCode" type="xs:int"/>
   <xs:element maxOccurs="unbounded" minOccurs="0" name="responses" nillable="false" type="ax21:Response"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="Response">
  <xs:sequence>
   <xs:element maxOccurs="unbounded" minOccurs="0" name="advisoryMessages" nillable="true" type="ax21:AdvisoryMessage"/>
   <xs:element minOccurs="0" name="apiVersion" nillable="false" type="xs:string"/>
   <xs:element minOccurs="0" name="offerList" nillable="true" type="ax21:OfferList"/>
   <xs:element maxOccurs="unbounded" minOccurs="0" name="profileRecord" nillable="true" type="ax21:NameValuePair"/>
   <xs:element minOccurs="0" name="sessionID" nillable="true" type="xs:string"/>
   <xs:element minOccurs="0" name="statusCode" type="xs:int"/>
```

```
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AdvisoryMessage">
   <xs:sequence>
    <xs:element minOccurs="0" name="detailMessage" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="message" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="messageCode" type="xs:int"/>
    <xs:element minOccurs="0" name="statusLevel" type="xs:int"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OfferList">
   <xs:sequence>
    <xs:element minOccurs="0" name="defaultString" nillable="true" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="recommendedOffers" nillable="true" type="ax21:Offer"/>
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Offer">
   <xs:sequence>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="additionalAttributes" nillable="true" type="ax21:NameValuePair"/>
    <xs:element minOccurs="0" name="description" nillable="true" type="xs:string"/>
    <xs:element maxOccurs="unbounded" minOccurs="0" name="offerCode" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="offerName" nillable="true" type="xs:string"/>
    <xs:element minOccurs="0" name="score" type="xs:int"/>
    <xs:element minOccurs="0" name="treatmentCode" nillable="true" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
 </xs:schema>
</wsdl:types>
<wsdl:message name="setAudienceRequest">
 <wsdl:part name="parameters" element="ns0:setAudience"/>
</wsdl:message>
<wsdl:message name="setAudienceResponse">
 <wsdl:part name="parameters" element="ns0:setAudienceResponse"/>
</wsdl:message>
<wsdl:message name="postEventRequest">
 <wsdl:part name="parameters" element="ns0:postEvent"/>
</wsdl:message>
<wsdl:message name="postEventResponse">
 <wsdl:part name="parameters" element="ns0:postEventResponse"/>
</wsdl:message>
<wsdl:message name="getOffersRequest">
 <wsdl:part name="parameters" element="ns0:getOffers"/>
</wsdl:message>
<wsdl:message name="getOffersResponse">
 <wsdl:part name="parameters" element="ns0:getOffersResponse"/>
</wsdl:message>
<wsdl:message name="startSessionRequest">
 <wsdl:part name="parameters" element="ns0:startSession"/>
</wsdl:message>
<wsdl:message name="startSessionResponse">
 <wsdl:part name="parameters" element="ns0:startSessionResponse"/>
</wsdl:message>
<wsdl:message name="getVersionRequest"/>
<wsdl:message name="getVersionResponse">
 <wsdl:part name="parameters" element="ns0:getVersionResponse"/>
</wsdl:message>
<wsdl:message name="setDebugRequest">
 <wsdl:part name="parameters" element="ns0:setDebug"/>
</wsdl:message>
<wsdl:message name="setDebugResponse">
 <wsdl:part name="parameters" element="ns0:setDebugResponse"/>
</wsdl:message>
<wsdl:message name="executeBatchRequest">
 <wsdl:part name="parameters" element="ns0:executeBatch"/>
</wsdl:message>
<wsdl:message name="executeBatchResponse">
 <wsdl:part name="parameters" element="ns0:executeBatchResponse"/>
</wsdl:message>
<wsdl:message name="getProfileRequest">
 <wsdl:part name="parameters" element="ns0:getProfile"/>
</wsdl:message>
<wsdl:message name="getProfileResponse">
 <wsdl:part name="parameters" element="ns0:getProfileResponse"/>
</wsdl:message>
<wsdl:message name="endSessionRequest">
 <wsdl:part name="parameters" element="ns0:endSession"/>
</wsdl:message>
<wsdl:message name="endSessionResponse">
 <wsdl:part name="parameters" element="ns0:endSessionResponse"/>
```

```
</wsdl:message>
<wsdl:portType name="InteractServicePortType">
 <wsdl:operation name="setAudience">
  <wsdl:input message="ns0:setAudienceRequest" wsaw:Action="urn:setAudience"/>
  <wsdl:output message="ns0:setAudienceResponse" wsaw:Action="urn:setAudienceResponse"/>
 </wsdl:operation>
 <wsdl:operation name="postEvent">
  <wsdl:input message="ns0:postEventRequest" wsaw:Action="urn:postEvent"/>
  <wsdl:output message="ns0:postEventResponse" wsaw:Action="urn:postEventResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getOffers">
  <wsdl:input message="ns0:getOffersRequest" wsaw:Action="urn:getOffers"/>
  <wsdl:output message="ns0:getOffersResponse" wsaw:Action="urn:getOffersResponse"/>
 </wsdl:operation>
 <wsdl:operation name="startSession">
  <wsdl:input message="ns0:startSessionRequest" wsaw:Action="urn:startSession"/>
  <wsdl:output message="ns0:startSessionResponse" wsaw:Action="urn:startSessionResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getVersion">
  <wsdl:input message="ns0:getVersionRequest" wsaw:Action="urn:getVersion"/>
  <wsdl:output message="ns0:getVersionResponse" wsaw:Action="urn:getVersionResponse"/>
 </wsdl:operation>
 <wsdl:operation name="setDebug">
  <wsdl:input message="ns0:setDebugRequest" wsaw:Action="urn:setDebug"/>
  <wsdl:output message="ns0:setDebugResponse" wsaw:Action="urn:setDebugResponse"/>
 </wsdl:operation>
 <wsdl:operation name="executeBatch">
  <wsdl:input message="ns0:executeBatchRequest" wsaw:Action="urn:executeBatch"/>
  <wsdl:output message="ns0:executeBatchResponse" wsaw:Action="urn:executeBatchResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getProfile">
  <wsdl:input message="ns0:getProfileRequest" wsaw:Action="urn:getProfile"/>
  <wsdl:output message="ns0:getProfileResponse" wsaw:Action="urn:getProfileResponse"/>
 </wsdl:operation>
 <wsdl:operation name="endSession">
  <wsdl:input message="ns0:endSessionRequest" wsaw:Action="urn:endSession"/>
  <wsdl:output message="ns0:endSessionResponse" wsaw:Action="urn:endSessionResponse"/>
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="InteractServiceSOAP11Binding" type="ns0:InteractServicePortType">
 <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="setAudience">
  <soap:operation soapAction="urn:setAudience" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="postEvent">
  <soap:operation soapAction="urn:postEvent" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getOffers">
  <soap:operation soapAction="urn:getOffers" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="startSession">
  <soap:operation soapAction="urn:startSession" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getVersion">
  <soap:operation soapAction="urn:getVersion" style="document"/>
```

```
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="setDebug">
  <soap:operation soapAction="urn:setDebug" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="executeBatch">
  <soap:operation soapAction="urn:executeBatch" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getProfile">
  <soap:operation soapAction="urn:getProfile" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="endSession">
  <soap:operation soapAction="urn:endSession" style="document"/>
  <wsdl:input>
   <soap:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="InteractServiceSOAP12Binding" type="ns0:InteractServicePortType">
 <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
 <wsdl:operation name="setAudience">
  <soap12:operation soapAction="urn:setAudience" style="document"/>
  <wsdl:input>
   <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap12:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="postEvent">
  <soap12:operation soapAction="urn:postEvent" style="document"/>
  <wsdl:input>
   <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap12:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getOffers">
  <soap12:operation soapAction="urn:getOffers" style="document"/>
  <wsdl:input>
   <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
   <soap12:body use="literal"/>
  </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="startSession">
  <soap12:operation soapAction="urn:startSession" style="document"/>
  <wsdl:input>
   <soap12:body use="literal"/>
  </wsdl:input>
  <wsdl:output>
```

```
    <soap12:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getVersion">
   <soap12:operation soapAction="urn:getVersion" style="document"/>
   <wsdl:input>
    <soap12:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap12:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="setDebug">
   <soap12:operation soapAction="urn:setDebug" style="document"/>
   <wsdl:input>
    <soap12:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap12:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="executeBatch">
   <soap12:operation soapAction="urn:executeBatch" style="document"/>
   <wsdl:input>
    <soap12:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap12:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getProfile">
   <soap12:operation soapAction="urn:getProfile" style="document"/>
   <wsdl:input>
    <soap12:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap12:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="endSession">
   <soap12:operation soapAction="urn:endSession" style="document"/>
   <wsdl:input>
    <soap12:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap12:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:binding name="InteractServiceHttpBinding" type="ns0:InteractServicePortType">
  <http:binding verb="POST"/>
  <wsdl:operation name="setAudience">
   <http:operation location="InteractService/setAudience"/>
   <wsdl:input>
    <mime:content part="setAudience" type="text/xml"/>
   </wsdl:input>
   <wsdl:output>
    <mime:content part="setAudience" type="text/xml"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="postEvent">
   <http:operation location="InteractService/postEvent"/>
   <wsdl:input>
    <mime:content part="postEvent" type="text/xml"/>
   </wsdl:input>
   <wsdl:output>
    <mime:content part="postEvent" type="text/xml"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getOffers">
   <http:operation location="InteractService/getOffers"/>
   <wsdl:input>
    <mime:content part="getOffers" type="text/xml"/>
   </wsdl:input>
   <wsdl:output>
    <mime:content part="getOffers" type="text/xml"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="startSession">
```

```
    <http:operation location="InteractService/startSession"/>
    <wsdl:input>
     <mime:content part="startSession" type="text/xml"/>
    </wsdl:input>
    <wsdl:output>
     <mime:content part="startSession" type="text/xml"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getVersion">
    <http:operation location="InteractService/getVersion"/>
    <wsdl:input>
     <mime:content part="getVersion" type="text/xml"/>
    </wsdl:input>
    <wsdl:output>
     <mime:content part="getVersion" type="text/xml"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="setDebug">
    <http:operation location="InteractService/setDebug"/>
    <wsdl:input>
     <mime:content part="setDebug" type="text/xml"/>
    </wsdl:input>
    <wsdl:output>
     <mime:content part="setDebug" type="text/xml"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="executeBatch">
    <http:operation location="InteractService/executeBatch"/>
    <wsdl:input>
     <mime:content part="executeBatch" type="text/xml"/>
    </wsdl:input>
    <wsdl:output>
     <mime:content part="executeBatch" type="text/xml"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="getProfile">
    <http:operation location="InteractService/getProfile"/>
    <wsdl:input>
     <mime:content part="getProfile" type="text/xml"/>
    </wsdl:input>
    <wsdl:output>
     <mime:content part="getProfile" type="text/xml"/>
    </wsdl:output>
   </wsdl:operation>
   <wsdl:operation name="endSession">
    <http:operation location="InteractService/endSession"/>
    <wsdl:input>
     <mime:content part="endSession" type="text/xml"/>
    </wsdl:input>
    <wsdl:output>
     <mime:content part="endSession" type="text/xml"/>
    </wsdl:output>
   </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="InteractService">
   <wsdl:port name="InteractServiceSOAP11port_http" binding="ns0:InteractServiceSOAP11Binding">
    <soap:address location="http://localhost:7001/interact/services/InteractService"/>
   </wsdl:port>
   <wsdl:port name="InteractServiceSOAP12port_http" binding="ns0:InteractServiceSOAP12Binding">
    <soap12:address location="http://localhost:7001/interact/services/InteractService"/>
   </wsdl:port>
   <wsdl:port name="InteractServiceHttpport" binding="ns0:InteractServiceHttpBinding">
    <http:address location="http://localhost:7001/interact/services/InteractService"/>
   </wsdl:port>
  </wsdl:service>
 </wsdl:definitions>
```

Chapter 13. Interact runtime environment configuration properties

This section describes all the configuration properties for the Interact runtime environment.

Interact | general

These configuration properties define general settings for your runtime environment environment, including the default logging level and the locale setting.

log4jConfig

Description

The location of the file containing the log4j properties. This path must be relative to the `INTERACT_HOME` environment variable. `INTERACT_HOME` is the location of the Interact installation directory.

Default value

`./conf/interact_log4j.properties`

asmUserForDefaultLocale

Description

The `asmUserForDefaultLocale` property defines the IBM Marketing Software user from which Interact derives its locale settings.

The locale settings define what language displays in the design time and what language advisory messages from the Interact API are in. If the locale setting does not match your machines operating system settings, Interact still functions, however the design time display and advisory messages may be in a different language.

Default value

`asm_admin`

Interact | general | learningTablesDataSource

These configuration properties define the data source settings for the built-in learning tables. You must define this data source if you are using Interact built-in learning.

If you create your own learning implementation using the Learning API, you can configure your custom learning implementation to read these values using the `ILearningConfig` interface.

jndiName

Description

Use this jndiName property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (Websphere or WebLogic) for the learning tables accessed by Interact runtime servers.

The learning tables are created by the aci_lrntab ddl file and contain the following tables (among others): UACI_AttributeValue and UACI_OfferStats.

Default value

No default value defined.

type

Description

The database type for the data source used by the learning tables accessed by the Interact runtime servers.

The learning tables are created by the aci_lrntab ddl file and contain the following tables (among others): UACI_AttributeValue and UACI_OfferStats.

Default value

SQLServer

Valid Values

SQLServer | DB2 | ORACLE

connectionRetryPeriod

Description

The ConnectionRetryPeriod property specifies the amount of time in seconds Interact automatically retries the database connection request on failure for the learning tables. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact will retry indefinitely; if the value is set to -1, no retry will be attempted.

The learning tables are created by the aci_lrntab ddl file and contain the following tables (among others): UACI_AttributeValue and UACI_OfferStats.

Default value

-1

connectionRetryDelay

Description

The ConnectionRetryDelay property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure for the learning tables. If the value is set to -1, no retry will be attempted.

The learning tables are created by the aci_lrntab ddl file and contain the following tables (among others): UACI_AttributeValue and UACI_OfferStats.

Default value

-1

schema

Description

The name of the schema containing the tables for the built-in learning module. Interact inserts the value of this property before all table names, for example, `UACI_IntChannel` becomes `schema.UACI_IntChannel`.

You do not have to define a schema. If you do not define a schema, Interact assumes that the owner of the tables is the same as the schema. You should set this value to remove ambiguity.

Default value

No default value defined.

Interact | general | prodUserDataSource

These configuration properties define the data source settings for the production profile tables. You must define this data source. This is the data source the runtime environment references when running interactive flowcharts after deployment.

jndiName

Description

Use this `jndiName` property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (Websphere or WebLogic) for the customer tables accessed by Interact runtime servers.

Default value

No default value defined.

type

Description

The database type for the customer tables accessed by Interact runtime servers.

Default value

SQLServer

Valid Values

SQLServer | DB2 | ORACLE

aliasPrefix

Description

The `AliasPrefix` property specifies the way Interact forms the alias name that Interact creates automatically when using a dimension table and writing to a new table in the customer tables accessed by Interact runtime servers..

Note that each database has a maximum identifier length; check the documentation for the database you are using to be sure that the value you set does not exceed the maximum identifier length for your database.

Default value

A

connectionRetryPeriod

Description

The ConnectionRetryPeriod property specifies the amount of time in seconds Interact automatically retries the database connection request on failure for the runtime customer tables. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact will retry indefinitely; if the value is set to -1, no retry will be attempted.

Default value

-1

connectionRetryDelay

Description

The ConnectionRetryDelay property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure for the Interact runtime customer tables. If the value is set to -1, no retry will be attempted.

Default value

-1

schema

Description

The name of the schema containing your profile data tables. Interact inserts the value of this property before all table names, for example, UACI_IntChannel becomes schema.UACI_IntChannel.

You do not have to define a schema. If you do not define a schema, Interact assumes that the owner of the tables is the same as the schema. You should set this value to remove ambiguity.

When you use a DB2 database, the schema name must be upper case.

Default value

No default value defined.

Interact | general | systemTablesDataSource

These configuration properties define the data source settings for the system tables for runtime environment. You must define this data source.

jndiName

Description

Use this jndiName property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (Websphere or WebLogic) for the runtime environment tables.

The runtime environment database is the database populated with the aci_runtime and aci_populate_runtime dll scripts and, for example, contains the following tables (among others): UACI_CHOfferAttrib and UACI_DefaultedStat.

Default value

No default value defined.

type

Description

The database type for the runtime environment system tables.

The runtime environment database is the database populated with the `aci_runtime` and `aci_populate_runtime` dll scripts and, for example, contains the following tables (among others): `UACI_CHOfferAttrib` and `UACI_DefaultedStat`.

Default value

SQLServer

Valid Values

SQLServer | DB2 | ORACLE

connectionRetryPeriod

Description

The `ConnectionRetryPeriod` property specifies the amount of time in seconds Interact automatically retries the database connection request on failure for the runtime system tables. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact will retry indefinitely; if the value is set to -1, no retry will be attempted.

The runtime environment database is the database populated with the `aci_runtime` and `aci_populate_runtime` dll scripts and, for example, contains the following tables (among others): `UACI_CHOfferAttrib` and `UACI_DefaultedStat`.

Default value

-1

connectionRetryDelay

Description

The `ConnectionRetryDelay` property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure for the Interact runtime system tables. If the value is set to -1, no retry will be attempted.

The runtime environment database is the database populated with the `aci_runtime` and `aci_populate_runtime` dll scripts and, for example, contains the following tables (among others): `UACI_CHOfferAttrib` and `UACI_DefaultedStat`.

Default value

-1

schema

Description

The name of the schema containing the tables for the runtime environment. Interact inserts the value of this property before all table names, for example, `UACI_IntChannel` becomes `schema.UACI_IntChannel`.

You do not have to define a schema. If you do not define a schema, Interact assumes that the owner of the tables is the same as the schema. You should set this value to remove ambiguity.

Default value

No default value defined.

Interact | general | systemTablesDataSource | loaderProperties

These configuration properties define the settings a database loader utility for the system tables for runtime environment. You need to define these properties if you are using a database loader utility only.

databaseName

Description

The name of the database the database loader connects to.

Default value

No default value defined.

LoaderCommandForAppend

Description

The `LoaderCommandForAppend` parameter specifies the command issued to invoke your database load utility for appending records to the contact and response history staging database tables in Interact. You need to set this parameter to enable the database loader utility for contact and response history data.

This parameter is specified as a full path name either to the database load utility executable or to a script that launches the database load utility. Using a script allows you to perform additional setup before invoking the load utility.

Most database load utilities require several arguments to be successfully launched. These can include specifying the data file and control file to load from and the database and table to load into. The tokens are replaced by the specified elements when the command is run.

Consult your database load utility documentation for the correct syntax to use when invoking your database load utility.

This parameter is undefined by default.

Tokens available to `LoaderCommandForAppend` are described in the following table.

Token	Description
<CONTROLFILE>	This token is replaced with the full path and filename to the temporary control file that Interact generates according to the template that is specified in the `LoaderControlFileTemplate` parameter.

Token	Description
<DATABASE>	This token is replaced with the name of the data source into which Interact is loading data. This is the same data source name used in the category name for this data source.
<DATAFILE>	This token is replaced with the full path and filename to the temporary data file created by Interact during the loading process. This file is in the Interact Temp directory, UNICA_ACTMPDIR.
<DBCOLUMNNUMBER>	This token is replaced with the column ordinal in the database.
<FIELDLENGTH>	This token is replaced with the length of the field being loaded into the database.
<FIELDNAME>	This token is replaced with the name of the field being loaded into the database.
<FIELDNUMBER>	This token is replaced with the number of the field being loaded into the database.
<FIELDTYPE>	This token is replaced with the literal "CHAR( )". The length of this field is specified between the (). If your database happens to not understand the field type, CHAR, you can manually specify the appropriate text for the field type and use the <FIELDLENGTH> token. For example, for SQLSVR and SQL2000 you would use "SQLCHAR(<FIELDLENGTH>)"
<NATIVETYPE>	This token is replaced with the type of database into which this field is loaded.
<NUMFIELDS>	This token is replaced with the number of fields in the table.
<PASSWORD>	This token is replaced with the database password from the current flowchart connection to the data source.
<TABLENAME>	This token is replaced with the database table name into which Interact is loading data.
<USER>	This token is replaced with the database user from the current flowchart connection to the data source.

Default value

No default value defined.

LoaderControlFileTemplateForAppend

Description

The LoaderControlFileTemplateForAppend property specifies the full path and filename to the control file template that has been previously configured in Interact. When this parameter is set, Interact dynamically builds a temporary control file based on the template that is specified here. The path and name of this temporary control file is available to the <CONTROLFILE> token that is available to the LoaderCommandForAppend property.

Before you use Interact in the database loader utility mode, you must configure the control file template that is specified by this parameter. The control file template supports the following tokens, which are dynamically replaced when the temporary control file is created by Interact.

See your database loader utility documentation for the correct syntax required for your control file. Tokens available to your control file template are the same as those for the LoaderControlFileTemplate property.

This parameter is undefined by default.

Default value

No default value defined.

LoaderDelimiterForAppend

Description

The LoaderDelimiterForAppend property specifies whether the temporary Interact data file is a fixed-width or delimited flat file, and, if it is delimited, the character or set of characters used as delimiters.

If the value is undefined, Interact creates the temporary data file as a fixed width flat file.

If you specify a value, it is used when the loader is invoked to populate a table that is not known to be empty. Interact creates the temporary data file as a delimited flat file, using the value of this property as the delimiter.

This property is undefined by default.

Default value

Valid Values

Characters, which you may enclose in double quotation marks, if desired.

LoaderDelimiterAtEndForAppend

Description

Some external load utilities require that the data file be delimited and that each line end with the delimiter. To accommodate this requirement, set the LoaderDelimiterAtEndForAppend value to TRUE, so that when the loader is invoked to populate a table that is not known to be empty, Interact uses delimiters at the end of each line.

Default value

FALSE

Valid Values

TRUE | FALSE

LoaderUseLocaleDP

Description

The LoaderUseLocaleDP property specifies, when Interact writes numeric values to files to be loaded by a database load utility, whether the locale-specific symbol is used for the decimal point.

Set this value to FALSE to specify that the period (.) is used as the decimal point.

Set this value to TRUE to specify that the decimal point symbol appropriate to the locale is used.

Default value

FALSE

Valid Values

TRUE | FALSE

Interact | general | testRunDataSource

These configuration properties define the data source settings for the test run tables for the Interact design environment. You must define this data source for at least one of your runtime environments. These are the tables used when you perform a test run of your interactive flowchart.

jndiName

Description

Use this jndiName property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (Websphere or WebLogic) for the customer tables accessed by the design environment when executing interactive flowcharts test runs.

Default value

No default value defined.

type

Description

The database type for the customer tables accessed by the design environment when executing interactive flowcharts test runs.

Default value

SQLServer

Valid Values

SQLServer | DB2 | ORACLE

aliasPrefix

Description

The AliasPrefix property specifies the way Interact forms the alias name that Interact creates automatically when using a dimension table and

writing to a new table for the customer tables accessed by the design environment when executing interactive flowcharts test runs.

Note that each database has a maximum identifier length; check the documentation for the database you are using to be sure that the value you set does not exceed the maximum identifier length for your database.

Default value

A

connectionRetryPeriod

Description

The `ConnectionRetryPeriod` property specifies the amount of time in seconds Interact automatically retries the database connection request on failure for the test run tables. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact will retry indefinitely; if the value is set to -1, no retry will be attempted.

Default value

-1

connectionRetryDelay

Description

The `ConnectionRetryDelay` property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure for the test run tables. If the value is set to -1, no retry will be attempted.

Default value

-1

schema

Description

The name of the schema containing the tables for interactive flowchart test runs. Interact inserts the value of this property before all table names, for example, `UACI_IntChannel` becomes `schema.UACI_IntChannel`.

You do not have to define a schema. If you do not define a schema, Interact assumes that the owner of the tables is the same as the schema. You should set this value to remove ambiguity.

Default value

No default value defined.

Interact | general | contactAndResponseHistoryDataSource

These configuration properties define the connection settings for the contact and response history data source required for the Interact cross-session response tracking. These settings are not related to the contact and response history module.

jndiName

Description

Use this jndiName property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (WebSphere or WebLogic) for the contact and response history data source required for the Interact cross-session response tracking.

Default value

type

Description

The database type for the data source used by the contact and response history data source required for the Interact cross-session response tracking.

Default value

SQLServer

Valid Values

SQLServer | DB2 | ORACLE

connectionRetryPeriod

Description

The ConnectionRetryPeriod property specifies the amount of time in seconds Interact automatically retries the database connection request on failure for the Interact cross-session response tracking. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact will retry indefinitely; if the value is set to -1, no retry will be attempted.

Default value

-1

connectionRetryDelay

Description

The ConnectionRetryDelay property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure for the Interact cross-session response tracking. If the value is set to -1, no retry will be attempted.

Default value

-1

schema

Description

The name of the schema containing the tables for the Interact cross-session response tracking. Interact inserts the value of this property before all table names, for example, UACI_IntChannel becomes schema.UACI_IntChannel.

You do not have to define a schema. If you do not define a schema, Interact assumes that the owner of the tables is the same as the schema. You should set this value to remove ambiguity.

Default value

No default value defined.

Interact | general | idsByType

These configuration properties define settings for ID numbers used by the contact and response history module.

initialValue

Description

The initial ID value used when generating IDs using the UACI_IDsByType table.

Default value

1

Valid Values

Any value greater than 0.

retries

Description

The number of retries before generating an exception when generating IDs using the UACI_IDsByType table.

Default value

20

Valid Values

Any integer greater than 0.

Interact | flowchart

This section defines configuration settings for interactive flowcharts.

defaultDateFormat

Description

The default date format used by Interact to convert Date to String and String to Date.

Default value

MM/dd/yy

idleFlowchartThreadTimeoutInMinutes

Description

The number of minutes Interact allows a thread dedicated to an interactive flowchart to be idle before releasing the thread.

Default value

5

idleProcessBoxThreadTimeoutInMinutes

Description

The number of minutes Interact allows a thread dedicated to an interactive flowchart process to be idle before releasing the thread.

Default value

5

maxSizeOfFlowchartEngineInboundQueue

Description

The maximum number of flowchart run requests Interact holds in queue. If this number of requests is reached, Interact will stop taking requests.

Default value

1000

maxNumberOfFlowchartThreads

Description

The maximum number of threads dedicated to interactive flowchart requests.

Default value

25

maxNumberOfProcessBoxThreads

Description

The maximum number of threads dedicated to interactive flowchart processes.

Default value

50

maxNumberOfProcessBoxThreadsPerFlowchart

Description

The maximum number of threads dedicated to interactive flowchart processes per flowchart instance.

Default value

3

minNumberOfFlowchartThreads

Description

The minimum number of threads dedicated to interactive flowchart requests.

Default value

10

minNumberOfProcessBoxThreads

Description

The minimum number of threads dedicated to interactive flowchart processes.

Default value

20

sessionVarPrefix

Description

The prefix for session variables.

Default value

SessionVar

Interact | flowchart | ExternalCallouts | [ExternalCalloutName]

This section defines the class settings for custom external callouts you have written with the external callout API.

class

Description

The name of the Java class represented by this external callout.

This is the Java class that you can access with the IBM Macro `EXTERNALCALLOUT`.

Default value

No default value defined.

classpath

Description

The classpath for the Java class represented by this external callout. The classpath must reference jar files on the runtime environment server. If you are using a server group and all runtime servers are using the same Marketing Platform, every server must have a copy of the jar file in the same location. The classpath must consist of absolute locations of jar files, separated by the path delimiter of the operating system of the runtime environment server, for example a semi-colon (;) on Windows and a colon (:) on UNIX systems. Directories containing class files are not accepted. For example, on a Unix system: `/path1/file1.jar:/path2/file2.jar`.

This classpath must be less than 1024 characters. You can use the manifest file in a .jar file to specify other .jar files so only one .jar file has to appear in your class path

This is the Java class that you can access with the IBM Macro `EXTERNALCALLOUT`.

Default value

No default value defined.

Interact | flowchart | ExternalCallouts | [ExternalCalloutName] | Parameter Data | [parameterName]

This section defines the parameter settings for a custom external callout you have written with the external callout API.

value

Description

The value for any parameter required by the class for the external callout.

Default value

No default value defined.

Example

If the external callout requires host name of an external server, create a parameter category named `host` and define the `value` property as the server name.

Interact | monitoring

This set of configuration properties enables you to define JMX monitoring settings. You need to configure these properties only if you are using JMX monitoring. There are separate JMX monitoring properties to define for the contact and response history module in the configuration properties for Interact design environment.

protocol

Description

Define the protocol for the Interact messaging service.

If you choose JMXMP you must include the following JAR files in your class path in order:

`Interact/lib/InteractJMX.jar;Interact/lib/jmxremote_optional.jar`

Default value

JMXMP

Valid Values

JMXMP | RMI

port

Description

The port number for the messaging service.

Default value

9998

enableSecurity

Description

A boolean which enables or disables JMXMP messaging service security for the Interact runtime server. If set to `true`, you must supply a user name and password to access the Interact runtime JMX service. This user credential is authenticated by the Marketing Platform for the runtime server. Jconsole does not allow empty password login.

This property has no effect if the protocol is RMI. This property has no effect on JMX for Campaign (the Interact design time).

Default value

True

Valid Values

True | False

Interact | monitoring | activitySubscribers

This set of configuration properties enables the root node for the settings that are related to remote subscribers that can receive periodic update on basic performance data in the Interact runtime environment.

heartbeatPeriodInSecs

Description

The interval in seconds when each runtime instance sends an update to subscribers.

Default value

60

Interact | monitoring | activitySubscribers | (target)

(target)

Description

The root node for the settings of a subscriber.

URL

Description

The URL of this subscriber. This endpoint must be able to accept JSON messages transported through HTTP.

continuousErrorsForAbort

Description

The number of continuous failed updates before the runtime instance stops sending more updates to this subscriber.

Default value

5

timeoutInMillis

Description

The time-out in milliseconds the send process times out during sending update to this subscriber.

Default value

1000

Valid Values

Enabled

Description

Whether this subscriber is enabled or disabled.

Default value

True

Valid Values

True or False

type

Description

The type of this data store. When this option is selected, the parameter **className** must be added with the value being the fully qualified name of this implementation class. **classPath** needs to be added with the URI of the JAR file if it is not in the class path of the Interact run time.

Default value

InteractLog

Valid Values

InteractLog, RelationalDB, and Custom

jmxInclusionCycles

Description

The interval in the multiplier of **heartbeatPeriodInSecs** that detailed JMX statistics are sent to this subscriber.

Default value

5

Valid Values

Interact | profile

This set of configuration properties control several of the optional offer serving features, including offer suppression and score override.

enableScoreOverrideLookup

Description

If set to True, Interact loads the score override data from the scoreOverrideTable when creating a session. If False, Interact does not load the marketing score override data when creating a session.

If true, you must also configure the Interact | profile | Audience Levels | (Audience Level) | scoreOverrideTable property. You need to define the scoreOverrideTable property for the audience levels you require only. Leaving the scoreOverrideTable blank for an audience level disables the score override table for the audience level.

Default value

False

Valid Values

True | False

enableOfferSuppressionLookup

Description

If set to `True`, Interact loads the offer suppression data from the `offerSuppressionTable` when creating a session. If `False`, Interact does not load the offer suppression data when creating a session.

If true, you must also configure the `Interact | profile | Audience Levels | (Audience Level) | offerSuppressionTable` property. You need to define the `enableOfferSuppressionLookup` property for the audience levels you require only.

Default value

`False`

Valid Values

True | False

enableProfileLookup

Description

In a new installation of Interact, this property is deprecated. In an upgraded installation of Interact, this property is valid until the first deployment.

The load behavior for a table used in an interactive flowchart but not mapped in the interactive channel. If set to `True`, Interact loads the profile data from the `profileTable` when creating a session.

If true, you must also configure the `Interact | profile | Audience Levels | (Audience Level) | profileTable` property.

The **Load this data in to memory when a visit session starts** setting in the interactive channel table mapping wizard overrides this configuration property.

Default value

`False`

Valid Values

True | False

defaultOfferUpdatePollPeriod

Description

The number of seconds the system waits before updating the default offers in the cache from the default offers table. If set to -1, the system doesn't update the default offers in the cache after the initial list is loaded into the cache when the runtime server starts.

Default value

-1

Interact | profile | Audience Levels | [AudienceLevelName]

This set of configuration properties enables you to define the table names required for additional Interact features. You are only required to define the table name if you are using the associated feature.

New category name

Description

The name of your audience level.

scoreOverrideTable

Description

The name of the table containing the score override information for this audience level. This property is applicable if you have set `enableScoreOverrideLookup` to true. You have to define this property for the audience levels for which you want to enable a score override table. If you have no score override table for this audience level, you can leave this property undefined, even if `enableScoreOverrideLookup` is set to true.

Interact looks for this table in the customer tables accessed by Interact runtime servers, defined by the `prodUserDataSource` properties.

If you have defined the `schema` property for this data source, Interact prepends this table name with the schema, for example, `schema.UACI_ScoreOverride`. If you enter a fully-qualified name, for example, `mySchema.UACI_ScoreOverride`, Interact does not prepend the schema name.

Default value

UACI_ScoreOverride

offerSuppressionTable

Description

The name of the table containing the offer suppression information for this audience level. You have to define this property for the audience levels for which you want to enable an offer suppression table. If you have no offer suppression table for this audience level, you can leave this property undefined. If `enableOfferSuppressionLookup` is set to true, this property must be set to a valid table.

Interact looks for this table in the customer tables accessed by runtime servers, defined by the `prodUserDataSource` properties.

Default value

UACI_BlackList

contactHistoryTable

Description

The name of the staging table for the contact history data for this audience level.

This table is stored in the runtime environment tables (`systemTablesDataSource`).

If you have defined the `schema` property for this data source, Interact prepends this table name with the schema, for example, `schema.UACI_CHStaging`. If you enter a fully-qualified name, for example, `mySchema.UACI_CHStaging`, Interact does not prepend the schema name.

If contact history logging is disabled, this property does not need to be set.

Default value

UACI_CHStaging

chOfferAttribTable

Description

The name of the contact history offer attributes table for this audience level.

This table is stored in the runtime environment tables (`systemTablesDataSource`).

If you have defined the `schema` property for this data source, Interact prepends this table name with the schema, for example, `schema.UACI_CHOfferAttrib`. If you enter a fully-qualified name, for example, `mySchema.UACI_CHOfferAttrib`, Interact does not prepend the schema name.

If contact history logging is disabled, this property does not need to be set.

Default value

UACI_CHOfferAttrib

responseHistoryTable

Description

The name of the response history staging table for this audience level.

This table is stored in the runtime environment tables (`systemTablesDataSource`).

If you have defined the `schema` property for this data source, Interact prepends this table name with the schema, for example, `schema.UACI_RHStaging`. If you enter a fully-qualified name, for example, `mySchema.UACI_RHStaging`, Interact does not prepend the schema name.

If response history logging is disabled, this property does not need to be set.

Default value

UACI_RHStaging

crossSessionResponseTable

Description

The name of the table for this audience level required for cross-session response tracking in the contact and response history tables accessible for the response tracking feature.

If you have defined the `schema` property for this data source, Interact prepends this table name with the schema, for example, `schema.UACI_XSessResponse`. If you enter a fully-qualified name, for example, `mySchema.UACI_XSessResponse`, Interact does not prepend the schema name.

If cross session response logging is disabled, this property does not need to be set.

Default value

UACI_XSessResponse

userEventLoggingTable

Description

This is the name of the database table that is used for logging user-defined event activities. Users defined events on the Events tab of the Interactive Channel summary pages in the Interact interface. The database table you specify here stores information such as the event ID, name, how many times this event occurred for this audience level since the last time the event activity cache was flushed, and so on.

If you have defined the schema property for this data source, Interact prepends this table name with the schema, for example, schema.UACI_UserEventActivity. If you enter a fully-qualified name, for example, mySchema.UACI_UserEventActivity, Interact does not prepend the schema name.

Default value

UACI_UserEventActivity

patternStateTable

Description

This is the name of the database table that is used for logging event pattern states, such as whether the pattern condition has been met or not, whether the pattern is expired or disabled, and so on.

If you have defined the schema property for this data source, Interact prepends this table name with the schema, for example, schema.UACI_EventPatternState. If you enter a fully-qualified name, for example, mySchema.UACI_EventPatternState, Interact does not prepend the schema name.

A patternStateTable is required for each audience level even if you do not use event patterns. The patternStateTable is based on the ddl of the included UACI_EventPatternState. The following is an example where the audience ID has two components; ComponentNum and ComponentStr.

```
CREATE TABLE UACI_EventPatternState_Composite
(
   UpdateTime bigint NOT NULL,
   State varbinary(4000),
   ComponentNum bigint NOT NULL,
   ComponentStr nvarchar(50) NOT NULL,
   CONSTRAINT PK_CustomerPatternState_Composite PRIMARY KEY
   (ComponentNum,ComponentStr,UpdateTime)
)
```

Default value

UACI_EventPatternState

Interact | profile | Audience Levels | [AudienceLevelName] | Offers by Raw SQL

This set of configuration properties enables you to define the table names required for additional Interact features. You are only required to define the table name if you are using the associated feature.

enableOffersByRawSQL

Description

If set to `True`, Interact enables the offersBySQL feature for this audience level that allows you to configure SQL code to be executed to create a desired set of candidate offers at runtime.. If `False`, Interact does not use the offersBySQL feature.

If you set this property to true, you may also configure the `Interact | profile | Audience Levels | (Audience Level) | Offers by Raw SQL | SQL Template` property to define one or more SQL templates.

Default value

`False`

Valid Values

True | False

cacheSize

Description

Size of cache used to store results of the OfferBySQL queries. Note that using a cache may have negative impact if query results are unique for most sessions.

Default value

`-1 (off)`

Valid Values

-1 | Value

cacheLifeInMinutes

Description

If the cache is enabled, this indicates the number of minutes before the system will clear the cache to avoid staleness.

Default value

`-1 (off)`

Valid Values

-1 | Value

defaultSQLTemplate

Description

The name of the SQL template to use if one is not specified via the API calls.

Default value

`None`

Valid Values

SQL template name

name

Configuration category

`Interact | profile | Audience Levels | [AudienceLevelName] | Offers by Raw SQL | (SQL Templates)`

Description

The name you want to assign to this SQL query template. Enter a descriptive name that will be meaningful when you use this SQL template in API calls. Note that if you use a name here that is *identical* to a name defined in the Interact List process box for an offerBySQL treatment, the SQL in the process box will be used rather than the SQL you enter here.

Default value

None

SQL

Configuration category

`Interact | profile | Audience Levels | [AudienceLevelName] | Offers by Raw SQL | (SQL Templates)`

Description

Contains the SQL query to be called by this template. The SQL query may contain references to variable names that are part of the visitor's session data (profile). For example, `select * from MyOffers where category = ${preferredCategory}` would rely on the session containing a variable named `preferredCategory`.

You should configure the SQL to query the specific offer tables you created during design time for use by this feature. Note that stored procedures are not supported here.

Default value

None

Interact | profile | Audience Levels | [AudienceLevelName] | SQL Template

These configuration properties let you define one or more SQL query templates used by the offersBySQL feature of Interact.

name

Description

The name you want to assign to this SQL query template. Enter a descriptive name that will be meaningful when you use this SQL template in API calls. Note that if you use a name here that is *identical* to a name defined in the Interact List process box for an offerBySQL treatment, the SQL in the process box will be used rather than the SQL you enter here.

Default value

None

SQL

Description

Contains the SQL query to be called by this template. The SQL query may contain references to variable names that are part of the visitor's session

data (profile). For example, `select * from MyOffers where category = ${preferredCategory}` would rely on the session containing a variable named `preferredCategory`.

You should configure the SQL to query the specific offer tables you created during design time for use by this feature. Note that stored procedures are not supported here.

Default value

None

Interact | profile | Audience Levels | [AudienceLevelName | Profile Data Services | [DataSource]

This set of configuration properties enables you to define the table names required for additional Interact features. You are only required to define the table name if you are using the associated feature. The Profile Data Services category provides information about a built-in data source (called Database) that is created for all audience levels, and which is pre-configured with a priority of 100. However, you can choose to modify or disable it. This category also contains a template for additional external data sources. When you click the template called **External Data Services** you can complete the configuration settings described here.

New category name

Description

(Not available for the default Database entry.) The name of the data source you are defining. The name you enter here must be unique among the data sources for the same audience level.

Default value

None

Valid Values

Any text string is allowed.

enabled

Description

If set to `True`, this data source is enabled for the audience level to which it is assigned. If `False`, Interact does not use this data source for this audience level.

Default value

`True`

Valid Values

True | False

className

Description

(Not available for the default Database entry.) The fully-qualified name of the data source class that implements `IInteractProfileDataService`.

Default value

None.

Valid Values

A string providing a fully-qualified class name.

classPath

Description

(Not available for the default Database entry.) An optional configuration setting providing the path to load this data source implementation class. If you omit it, the class path of the containing application server is used by default.

Default value

Not shown, but the class path of the containing application server is used by default if no value is provided here.

Valid Values

A string providing the class path.

priority

Description

The priority of this data source within this audience level. It has to be a unique value among all of the data sources for each audience level. (That is, if a priority is set to 100 for a data source, no other data source within the audience level may have a priority of 100.)

Default value

`100` for the default Database, `200` for user-defined data source

Valid Values

Any non-negative integer is allowed.

Interact | offerserving

These configuration properties define the generic learning configuration properties. If you are using built-in learning, to tune your learning implementation, use the configuration properties for the design environment.

offerTieBreakMethod

Description

The `offerTieBreakMethod` property defines the behavior of offer serving when two offers have equivalent (tied) scores. If you set this property to its default value of Random, Interact presents a random choice from among the offers that have equivalent scores. If you set this configuration to Newer Offer, Interact serves up the newer offer (based on having a higher offer ID) ahead of the older offer (lower offer ID) in the case where the scores among the offers are the same.

Note:

Interact has an optional feature that allows the administrator to configure the system to return the offers in random order independent of the score, by setting the `percentRandomSelection` option (`Campaign | partitions | [partition_number] | Interact | learning | percentRandomSelection`).

The offerTieBreakMethod property described here is used only when percentRandomSelection is set to zero (disabled).

Default value

Random

Valid Values

Random | Newer Offer

optimizationType

Description

The optimizationType property defines whether Interact uses a learning engine to assist with offer assignments. If set to NoLearning, Interact does not use learning. If set to BuiltInLearning, Interact uses the Bayesian learning engine built with Interact. If set to ExternalLearning, Interact uses a learning engine you provide. If you select ExternalLearning, you must define the externalLearningClass and externalLearningClassPath properties.

Default value

NoLearning

Valid Values

NoLearning | BuiltInLearning | ExternalLearning

segmentationMaxWaitTimeInMS

Description

The maximum number of milliseconds that the runtime server waits for an interactive flowchart to complete before getting offers.

Default value

5000

treatmentCodePrefix

Description

The prefix prepended to treatment codes.

Default value

No default value defined.

effectiveDateBehavior

Description

Determines whether Interact should use an offer's effective date in filtering out offers that are presented to a visitor. Values include:

- -1 tells Interact to ignore the effective date on the offer.

  0 tells Interact to use the effective date to filter the offer, so that if the offer effective date is earlier than or equal to the current date, the offer effective date, the offer is served to visitors.

  If there is an **effectiveDateGracePeriod** value set, the grace period is also applied to determine whether to serve the offer.

- Any positive integer tells Interact to use the current date plus the value of this property to determine whether to serve the offer to visitors, so that if the offer effective date is earlier than the current date plus the value of this property, the offer is served to visitors.

  If there is an **effectiveDateGracePeriod** value set, the grace period is also applied to determine whether to serve the offer.

Default value

-1

effectiveDateGracePeriodOfferAttr

Description

Specifies the name of the custom attribute in an offer definition that indicates the effective date grace period. For example, you might configure this property with a value of `AltGracePeriod`. You would then define offers with a custom attribute called `AltGracePeriod` that is used to specify the number of days to use as a grace period with the **effectiveDateBehavior** property.

Suppose you create a new offer template with an effective date of 10 days from the current date, and include a custom attribute called `AltGracePeriod`. When you create an offer using the template, if you set the value of `AltGracePeriod` to 14 days, the offer would be served to visitors, because the current date is within the grace period of the offer effective date.

Default value

Blank

alwaysLogLearningAttributes

Description

Indicates whether Interact should write information about visitor attributes used by the learning module to the log files. Note that settings this value to true may affect learning performance and log file sizes.

Default value

False

Interact | offerserving | Built-in Learning Config

These configuration properties define the database write settings for built-in learning. To tune your learning implementation, use the configuration properties for the design environment.

version

Description

You can select 1 or 2. Version 1 is the basic configuration version that does not use parameters to set thread and record limits. Version 2 is the enhanced configuration version that lets you set thread and record parameter to improve performance. These parameters perform aggregation and deletion when these parameter limits are reached.

Default value

1

insertRawStatsIntervalInMinutes

Description

The number of minutes the Interact learning module waits before inserting more rows into the learning staging tables. You may need to modify this time based on the amount of data the learning module is processing in your environment.

Default value

5

Valid Values

A positive integer

aggregateStatsIntervalInMinutes

Description

The number of minutes the Interact learning module waits between aggregating data in the learning stats tables. You may need to modify this time based on the amount of data the learning module is processing in your environment.

Default value

15

Valid Values

An integer greater than zero.

autoAdjustPercentage

Description

The value that determines the percentage of data the run of aggregation tries to process based on the metrics of the previous run. By default, this value is set to zero, which means the aggregator processes all staging records, and this auto adjustment functionality is disabled.

Default value

0

Valid Values

A number between 0 and 100.

enableObservationModeOnly

Description

If set to `True`, enables a learning mode where Interact collects data for learning without using that data for recommendations or offer arbitration. This allows you to operate self-learning in a startup mode until you determine that enough data is collected for recommendations.

Default value

False

Valid Values

True | False

excludeAbnormalAttribute

Description

The setting that determines whether to mark those attributes as invalid. If set to `IncludeAttribute`, abnormal attributes are included not marked as invalid. If set to `ExcludeAttribute`, abnormal attributes are excluded and marked as invalid.

Default value

IncludeAttribute

Valid Values

IncludeAttribute | ExcludeAttribute

saveOriginalValues

Description

You can set the values as “All Values”, “Binned Values”, or “None”. This will control what values will be logged in table UACI_LearningAttributeHist.

If “All Values” is selected then all learning attributes will be logged in the table. If this parameter is set to “Binned Values” then only those attributes will be logged in the table for which bins are created under “Interact-> Global Learning”.

If set to “None” no values will be logged in UACI_LearningAttributeHist .

By default this is set to “None”.

Default value

None

Valid Values

All Values | Binned Values | None

Interact | offerserving | Built-in Learning Config | Parameter Data | [parameterName]

These configuration properties define any parameters for your external learning module.

numberOfThreads

Description

The maximum number of threads the learning aggregator uses to process the data. A valid value is a positive integer, and should not be more than the maximum number of connections that are configured in the learning data source. This parameter is used only by aggregator version 2.

Default value

10

maxLogTimeSpanInMin

Description

If aggregator version 1 is selected, you can process the staging records in iterations to avoid overly large database batches. In this case, those staging

records are processed by chunks; iteration by iteration in a single aggregation cycle. The value of this parameter specifies the maximum time span of staging records the aggregator tries to process in each iteration. This time span is based on LogTime field that is associated to each staging record, and only the records whose LogTime falls into the earliest time window is processed. A valid value is an integer that is not negative. If the value is 0, there is no limit, which means all the staging records are processed in a single iteration.

Default value

0

maxRecords

Description

If aggregator version 2 is selected, you can process the staging records in iterations to avoid overly large database batches. In this case, those staging records are processed in chunks; iteration by iteration in a single aggregation cycle. The value of this parameter specifies the maximum number of staging records the aggregator tries to process in each iteration. A valid value is an integer that is not negative. If the value is 0, there is no limit, which means all the staging records are processed in a single iteration.

Default value

0

value

Description

The value for any parameter that is required by the class for a built-in learning module.

Default value

No default value defined.

Interact | offerserving | External Learning Config

These configuration properties define the class settings for an external learning module you wrote using the learning API.

class

Description

If `optimizationType` is set to `ExternalLearning`, set `externalLearningClass` to the class name for the external learning engine.

Default value

No default value defined.

Availability

This property is applicable only if `optimizationType` is set to `ExternalLearning`.

classPath

Description

If optimizationType is set to ExternalLearning, set externalLearningClass to the classpath for the external learning engine.

The classpath must reference jar files on the runtime environment server. If you are using a server group and all runtime servers are using the same Marketing Platform, every server must have a copy of the jar file in the same location. The classpath must consist of absolute locations of jar files, separated by the path delimiter of the operating system of the runtime environment server, for example a semi-colon (;) on Windows and a colon (:) on UNIX systems. Directories containing class files are not accepted. For example, on a Unix system: /path1/file1.jar:/path2/file2.jar.

This classpath must be less than 1024 characters. You can use the manifest file in a .jar file to specify other .jar files so only one .jar file has to appear in your class path

Default value

No default value defined.

Availability

This property is applicable only if optimizationType is set to ExternalLearning.

Interact | offerserving | External Learning Config | Parameter Data | [parameterName]

These configuration properties define any parameters for your external learning module.

value

Description

The value for any parameter required by the class for an external learning module.

Default value

No default value defined.

Example

If the external learning module requires a path to an algorithm solver application, you would create a parameter category called solverPath and define the value property as the path to the application.

Interact | offerserving | Constraints

These configuration properties define the constraints placed upon the offer serving process.

maxOfferAllocationInMemoryPerInstance

Description

The size of a block of offers. Interact keeps a pool of offers in memory so that the system does not have to query to database each time an offer is returned. Every time an offer is returned, the pool is adjusted. When the pool is exhausted, Interact gets another block of offers to fill the pool.

Default value

1000

Valid Values

An integer greater than 0.

maxDistributionPerIntervalPerInstanceFactor

Description

The constraint percentage for a given offer allocation for a runtime server to support the distribution across runtime servers.

Default value

100

Valid Values

An integer between 0 and 100.

constraintCleanupIntervalInDays

Description

How often the disabled counts from the UACI_OfferCount table are cleaned up. A value less than 1 disables this feature.

Default value

7

Valid Values

An integer greater than 0.

Interact | services

The configuration properties in this category define settings for all the services which manage collecting contact and response history data and statistics for reporting and writing to the runtime environment system tables.

externalLoaderStagingDirectory

Description

This property defines the location of the staging directory for a database load utility.

Default value

No default value defined.

Valid Values

A path relative to the Interact installation directory or an absolute path to a staging directory.

If you enable a database load utility, you must set the `cacheType` property in the `contactHist` and `responstHist` categories to `External Loader File`.

Interact | services | contactHist

The configuration properties in this category define the settings for the service that collects data for the contact history staging tables.

enableLog

Description

If `true`, enables the service which collects data for recording the contact history data. If `false`, no data is collected.

Default value

True

Valid Values

True | False

cacheType

Description

Defines whether the data collected for contact history is kept in memory (`Memory Cache`) or in a file (`External Loader file`). You can use `External Loader File` only if you have configured Interact to use a database loader utility.

If you select `Memory Cache`, use the `cache` category settings. If you select `External Loader File`, use the `fileCache` category settings.

Default value

Memory Cache

Valid Values

Memory Cache | External Loader File

Interact | services | contactHist | cache

The configuration properties in this category define the cache settings for the service that collects data for the contact history staging table.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected contact history data to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | contactHist | contactStatusCodes

The configuration properties in this category defines the settings for the custom contact status type which can be passed into Interact together with contact events..

New category name

Description

This property defines the name of contact status code category.

Code

Description

This property defines the custom code for your contact type. This defined code must exist in IBM Campaign system table UA_ContactStatus.

action

Description

The action corresponding to the custom contact type code. The action defined here will override the action defined for in the IBM Campaign System table UA_ContactStatus..

Default value

None

Valid value

LogContact | None

Interact | services | contactHist | fileCache

The configuration properties in this category define the cache settings for the service that collects contact history data if you are using a database loader utility.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected contact history data to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | defaultedStats

The configuration properties in this category define the settings for the service that collects the statistics regarding the number of times the default string for the interaction point was used.

enableLog

Description

If `true`, enables the service that collects the statistics regarding the number of times the default string for the interaction point was used to the `UACI_DefaultedStat` table. If `false`, no default string statistics are collected.

If you are not using IBM reporting, you can set this property to `false` since the data collection is not required.

Default value

True

Valid Values

True | False

Interact | services | defaultedStats | cache

The configuration properties in this category define the cache settings for the service that collects the statistics regarding the number of times the default string for the interaction point was used.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected default string statistics to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | eligOpsStats

The configuration properties in this category define the settings for the service that writes the statistics for eligible offers.

enableLog

Description

If `true`, enables the service that collects the statistics for eligible offers. If `false`, no eligible offer statistics are collected.

If you are not using IBM reporting, you can set this property to `false` since the data collection is not required.

Default value

True

Valid Values

True | False

Interact | services | eligOpsStats | cache

The configuration properties in this category define the cache settings for the service that collects the eligible offer statistics.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected eligible offer statistics to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | eventActivity

The configuration properties in this category define the settings for the service that collects the event activity statistics.

enableLog

Description

If `true`, enables the service that collects the event activity statistics. If `false`, no event statistics are collected.

If you are not using IBM reporting, you can set this property to `false` since the data collection is not required.

Default value

True

Valid Values

True | False

Interact | services | eventActivity | cache

The configuration properties in this category define the cache settings for the service that collects the event activity statistics.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected event activity statistics to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | eventPattern

The configuration properties in the `eventPattern` category define the settings for the service that collects the event pattern activity statistics.

persistUnknownUserStates

Description

Determines whether the event pattern states for an unknown audience ID (visitor) is retained in the database. By default, when a session ends, the statuses of all the updated event patterns associated with the visitor's audience ID are stored in the database, provided that the audience ID is known (that is, the visitor's profile can be found in the profile data source).

The `persistUnknownUserStates` property determines what happens if the audience ID is not known. By default, this property is set to `False`, and for unknown audience IDs, the event pattern states are discarded at the end of the session.

If you set this property to `True`, the event pattern states of unknown users (whose profile cannot be find in the configured profile data service) will be persisted.

Default value

False

Valid Values

True | False

mergeUnknowUserInSessionStates

Description

Determines how the event pattern states for unknown audience IDs (visitors) are retained. If the audience ID switches in the middle of a session, Interact tries to load the saved event pattern states for the new audience ID from the database table. When the audience ID was unknown previously, and you set the `mergeUnknowUserInSessionStates` property is to `True`, the user event activities belonging to the previous audience ID in the same session will be merged into the new audience ID.

Default value

False

Valid Values

True | False

enableUserEventLog

Description

Determines whether user event activities are logged in the database.

Default value

False

Valid Values

True | False

Interact | services | eventPattern | userEventCache

The configuration properties in the `userEventCache` category define the settings that determine when event activity is moved from the cache to persist in the database.

threshold

Description

Determines the maximum number of event pattern states that can be stored in the event pattern state cache. When the limit is reached, the least-recently used states are flushed from the cache.

Default value

100

Valid Values

The desired number of event pattern states to retain in the cache.

insertPeriodInSecs

Description

Determines the maximum length of time in seconds that user event activities are queued in memory. When the time limit specified by this property is reached, those activities are persisted into the database.

Default value

3600 (60 minutes)

Valid Values

The desired number of seconds.

Interact | services | eventPattern | advancedPatterns

The configuration properties in this category control whether integration with Interact Advanced Patterns is enabled, and they define the timeout intervals for connections with Interact Advanced Patterns.

enableAdvancedPatterns

Description

If `true`, enables integration with Interact Advanced Patterns. If `false`, integration is not enabled. If integration was previously enabled, Interact uses the most recent pattern states received from Interact Advanced Patterns.

Default value

True

Valid Values

True | False

connectionTimeoutInMilliseconds

Description

Maximum time it can take to make an HTTP connection from the Interact real time environment to Interact Advanced Patterns. If the request times out, Interact uses the last saved data from patterns.

Default value

30

readTimeoutInMilliseconds

Description

After an HTTP connection is established between the Interact real time environment and Interact Advanced Patterns,and a request is sent to the Interact Advanced Patterns to get the status of an event pattern, the maximum time it can take to receive data. If the request times out, Interact uses the last saved data from patterns.

Default value

100

connectionPoolSize

Description

Size of the HTTP connection pool for communication between the Interact real time environment and Interact Advanced Patterns.

Default value

10

Interact | services | eventPattern | advancedPatterns | autoReconnect

The configuration properties in this category specify parameters for the automatic reconnection feature in the integration with Interact Advanced Patterns.

enable

Description

Determines whether the system to reconnects automatically if connection problems occur between the Interact real time environment and Interact Advanced Patterns. The default value of **True** enables this feature.

Default value

True

Valid Values

True | False

durationInMinutes

Description

This property specifies the time interval, in minutes, during which the system to evaluates repeated connection problems occurring between the Interact real time environment and Interact Advanced Patterns.

Default value

10

numberOfFailuresBeforeDisconnect

Description

This property specifies the number of connection failures allowed during the specified time period before the system automatically disconnects from Interact Advanced Patterns.

Default value

3

consecutiveFailuresBeforeDisconnect

Description

Determines whether the automatic reconnection feature evaluates only consecutive failures of the connection between the Interact real time environment with Interact Advanced Patterns. If you set this value to **False**, all failures within the specified time interval are evaluated.

Default value

True

sleepBeforeReconnectDurationInMinutes

Description

The system waits the number of minutes specified in this property before reconnecting after the system disconnects due to repeated failures as specified in the other properties in this category.

Default value

5

sendNotificationAfterDisconnect

Description

This property determines whether the system sends an email notification when a connection failure occurs. The notification message includes the Interact real time instance name for which failure occurred and the amount of time before reconnection occurs, as specified in the **sleepBeforeReconnectDurationInMinutes** property. The default value of **True** means that notifications are sent.

Default value

True

Interact | services | customLogger

The configuration properties in this category define the settings for the service that collects custom data to write to a table (an event which uses the `UACICustomLoggerTableName` event parameter).

enableLog

Description

If `true`, enables the custom log to table feature. If `false`, the `UACICustomLoggerTableName` event parameter has no effect.

Default value

True

Valid Values

True | False

Interact | services | customLogger | cache

The configuration properties in this category define the cache settings for the service that collects custom data to a table (an event which uses the UACICustomLoggerTableName event parameter).

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected custom data to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | responseHist

The configuration properties in this category define the settings for the service that writes to the response history staging tables.

enableLog

Description

If true, enables the service that writes to the response history staging tables. If false, no data is written to the response history staging tables.

The response history staging table is defined by the responseHistoryTable property for the audience level. The default is UACI_RHStaging.

Default value

True

Valid Values

True | False

cacheType

Description

Defines whether the cache is kept in memory or in a file. You can use External Loader File only if you configured Interact to use a database loader utility.

If you select Memory Cache, use the cache category settings. If you select External Loader File, use the fileCache category settings.

Default value

Memory Cache

Valid Values

Memory Cache | External Loader File

actionOnOrphan

Description

This setting determines what to do with response events that do not have corresponding contact events. If set to `NoAction`, the response event is processed as if the corresponding contact event was posted. If set to `Warning`, the response event is processed as if the corresponding contact event was posted, but a warning message is written into `interact.log`. If set to `Skip`, the response even is not processed, and an error message is written into `interact.log`. The setting that you choose here is effective regardless if response history logging is enabled.

Default value

NoAction

Valid Values

NoAction | Warning | Skip

Interact | services | responseHist | cache

The configuration properties in this category define the cache settings for the service that collects the response history data.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected response history data to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | response Hist | responseTypeCodes

The configuration properties in this category define the settings for the response history service.

New category name

Description

The name of your response type code.

code

Description

The custom code for your response type.

Default value

The custom code added in the UA_UsrResponseType table.

action

Description

The action corresponding to the custom response type code.

The action defined for the event this is posted overrides the action defined here. Therefore, if a logAccept event is posted without responseTypeCode, this event is treated as an acceptance event. If a logAccept event is posted with a responseTypeCode that exists in this configuration, the configured action is used to determine if it is an acceptance event. If a logAccept event is posted with a responseTypeCode that does not exist in this configuration, this event is not treated as an acceptance event. When an event is treated as an acceptance event, the learning statistics are updated accordingly if learning is enabled. Offer expression rules are evaluated if there is one based on the acceptance of this offer.

Default value

None

Valid Values

LogAccept | LogReject | None

Interact | services | responseHist | fileCache

The configuration properties in this category define the cache settings for the service that collects the response history data if you are using a database loader utility.

threshold

Description

The number of records accumulated before Interact writes them to the database.

`responseHist` - The table defined by the `responseHistoryTable` property for the audience level. The default is `UACI_RHStaging`.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the database.

Default value

3600

Interact | services | crossSessionResponse

The configuration properties in this category define general settings for the crossSessionResponse service and the xsession process. You only need to configure these settings if you are using Interact cross-session response tracking.

enableLog

Description

If `true`, enables the crossSessionResponse service and Interact writes data to the cross-session response tracking staging tables. If `false`, disables the crossSessionResponse service.

Default value

False

xsessionProcessIntervalInSecs

Description

The number of seconds between runs of the xsession process. This process moves data from the cross-session response tracking staging tables to the response history staging table and the built-in learning module.

Default value

180

Valid Values

An integer greater than zero

purgeOrphanResponseThresholdInMinutes

Description

The number of minutes the crossSessionResponse service waits before marking any responses that do not match contacts in the contact and response history tables.

If a response has no match in the contact and response history tables, after `purgeOrphanResponseThresholdInMinutes` minutes, Interact marks the response with a value of -1 in the `Mark` column of the xSessResponse staging table. You can then manually match or delete these responses.

Default value

180

Interact | services | crossSessionResponse | cache

The configuration properties in this category define the cache settings for the service that collects cross-session response data.

threshold

Description

The number of records accumulated before the flushCacheToDB service writes the collected cross-session response data to the database.

Default value

100

insertPeriodInSecs

Description

The number of seconds between forced writes to the `XSessResponse` table.

Default value

3600

Interact | services | crossSessionResponse | OverridePerAudience | [AudienceLevel] | TrackingCodes | byTreatmentCode

The properties in this section define how cross-session response tracking matches treatment codes to contact and response history.

SQL

Description

This property defines whether Interact uses the System Generated SQL or custom SQL defined in the `OverrideSQL` property.

Default value

Use System Generated SQL

Valid Values

Use System Generated SQL | Override SQL

OverrideSQL

Description

If you do not use the default SQL command to match the treatment code to the contact and response history, enter the SQL or stored procedure here.

This value is ignored if `SQL` is set to `Use System Generated SQL`.

Default value

useStoredProcedure

Description

If set to true, the `OverrideSQL` must contain a reference to a stored procedure which matches the treatment code to the contact and response history.

If set to false, the `OverrideSQL`, if used, must be an SQL query.

Default value

false

Valid Values

true | false

Type

Description

The associated TrackingCodeType defined in the `UACI_TrackingType` table in the runtime environment tables. Unless you revise the `UACI_TrackingType` table, the `Type` must be 1.

Default value

1

Valid Values

An integer defined in the `UACI_TrackingType` table.

Interact | services | crossSessionResponse | OverridePerAudience | [AudienceLevel] | TrackingCodes | byOfferCode

The properties in this section define how cross-session response tracking matches offer codes to contact and response history.

SQL

Description

This property defines whether Interact uses the System Generated SQL or custom SQL defined in the `OverrideSQL` property.

Default value

Use System Generated SQL

Valid Values

Use System Generated SQL | Override SQL

OverrideSQL

Description

If you do not use the default SQL command to match the offer code to the contact and response history, enter the SQL or stored procedure here.

This value is ignored if `SQL` is set to `Use System Generated SQL`.

Default value

useStoredProcedure

Description

If set to true, the `OverrideSQL` must contain a reference to a stored procedure which matches the offer code to the contact and response history.

If set to false, the `OverrideSQL`, if used, must be an SQL query.

Default value

false

Valid Values

true | false

Type

Description

The associated TrackingCodeType defined in the `UACI_TrackingType` table in the runtime environment tables. Unless you revise the `UACI_TrackingType` table, the `Type` must be 2.

Default value

2

Valid Values

An integer defined in the `UACI_TrackingType` table.

Interact | services | crossSessionResponse | OverridePerAudience | [AudienceLevel] | TrackingCodes | byAlternateCode

The properties in this section define how cross-session response tracking matches a user-defined alternate code to contact and response history.

Name

Description

This property defines the name for the alternate code. This must match the Name value in the UACI_TrackingType table in the runtime environment tables.

Default value

OverrideSQL

Description

The SQL command or stored procedure to match the alternate code to the contact and response history by offer code or treatment code.

Default value

useStoredProcedure

Description

If set to true, the OverrideSQL must contain a reference to a stored procedure which matches the alternate code to the contact and response history.

If set to false, the OverrideSQL, if used, must be an SQL query.

Default value

false

Valid Values

true | false

Type

Description

The associated TrackingCodeType defined in the UACI_TrackingType table in the runtime environment tables.

Default value

3

Valid Values

An integer defined in the UACI_TrackingType table.

Interact | services | threadManagement | contactAndResponseHist

The configuration properties in this category define thread management settings for the services which collect data for the contact and response history staging tables.

corePoolSize

Description

The number of threads to keep in the pool, even if they are idle, for collecting the contact and response history data.

Default value

5

maxPoolSize

Description

The maximum number of threads to keep in the pool for collecting the contact and response history data.

Default value

5

keepAliveTimeSecs

Description

When the number of threads is greater than the core, this is the maximum time that excess idle threads will wait for new tasks before terminating for collecting the contact and response history data.

Default value

5

queueCapacity

Description

The size of the queue used by the thread pool for collecting the contact and response history data.

Default value

1000

termWaitSecs

Description

At the shutdown of the runtime server, this is the number of seconds to wait for service threads to complete collecting the contact and response history data.

Default value

5

Interact | services | threadManagement | allOtherServices

The configuration properties in this category define the thread management settings for the services which collect the offer eligibility statistics, event activity statistics, default string usage statistics, and the custom log to table data.

corePoolSize

Description

The number of threads to keep in the pool, even if they are idle, for the services which collect the offer eligibility statistics, event activity statistics, default string usage statistics, and the custom log to table data.

Default value

5

maxPoolSize

Description

The maximum number of threads to keep in the pool for the services which collect the offer eligibility statistics, event activity statistics, default string usage statistics, and the custom log to table data.

Default value

5

keepAliveTimeSecs

Description

When the number of threads is greater than the core, this is the maximum time that excess idle threads wait for new tasks before terminating for the services which collect the offer eligibility statistics, event activity statistics, default string usage statistics, and the custom log to table data.

Default value

5

queueCapacity

Description

The size of the queue used by the thread pool for the services which collect the offer eligibility statistics, event activity statistics, default string usage statistics, and the custom log to table data.

Default value

1000

termWaitSecs

Description

At the shutdown of the runtime server, this is the number of seconds to wait for service threads to complete for the services which collect the offer eligibility statistics, event activity statistics, default string usage statistics, and the custom log to table data.

Default value

5

Interact | services | threadManagement | flushCacheToDB

The configuration properties in this category define the thread management settings for the threads that write collected data in cache to the runtime environment database tables.

corePoolSize

Description

The number of threads to keep in the pool for scheduled threads that write cached data to the data store.

Default value

5

maxPoolSize

Description

The maximum number of threads to keep in the pool for scheduled threads that that write cached data to the data store.

Default value

5

keepAliveTimeSecs

Description

When the number of threads is greater than the core, this is the maximum time that excess idle threads wait for new tasks before terminating for scheduled threads that that write cached data to the data store.

Default value

5

queueCapacity

Description

The size of the queue used by the thread pool for scheduled threads that that write cached data to the data store.

Default value

1000

termWaitSecs

Description

At the shutdown of the runtime server, this is the number of seconds to wait for service threads to complete for scheduled threads that that write cached data to the data store.

Default value

5

Interact | services | threadManagement | eventHandling

The configuration properties in this category define the thread management settings for the services which collect data for event handling.

corePoolSize

Description

The number of threads to keep in the pool, even if they are idle, for collecting event handling data.

Default value

1

maxPoolSize

Description

The maximum number of threads to keep in the pool for the services which collect the event handling data.

Default value

5

keepAliveTimeSecs

Description

When the number of threads is greater than the core, this is the maximum time that excess idle threads wait for new tasks before terminating for collecting the event handling data.

Default value

5

queueCapacity

Description

The size of the queue used by the thread pool for collecting event handling data.

Default value

1000

termWaitSecs

Description

At the shutdown of the runtime server, this is the number of seconds to wait for service threads to complete for the services which collect the event handling data.

Default value

5

Interact | services | configurationMonitor

The configuration properties in this category allow you to enable or disable integration with Interact Advanced Patterns without having to restart Interact real time, and they define the interval for polling the property value that enables the integration.

enable

Description

If `true`, enables the service that refreshes the value of the **Interact | services | eventPattern | advancedPatterns enableAdvancedPatterns**

property. If `false`, you must restart Interact real time when you change the value of the **Interact | services | eventPattern | advancedPatterns enableAdvancedPatterns** property.

Default value

False

Valid Values

True | False

refreshIntervalInMinutes

Description

Defines the time interval for polling the value of the **Interact | services | eventPattern | advancedPatterns enableAdvancedPatterns** property.

Default value

5

Interact | cacheManagement

This set of configuration properties defines settings for selecting and configuring each of the supported cache managers that you can use to improve the performance of Interact, such as EHCache, which is built-in to your Interact installationWebSphere eXtreme Scale caching, which is an optional add-on, or another external caching system.

Use the **Interact | cacheManagement | Cache Managers** configuration properties to configure the cache manager you want to use. Use the **Interact | cacheManagement | caches** configuration properties to specify which cache manager Interact should use to improve performance.

Interact | cacheManagement | Cache Managers

The Cache Managers category specifies the parameters for the cache management solutions you plan to use with Interact.

Interact | cacheManagement | Cache Managers | EHCache

The EHCache category specifies the parameters for the EHCache cache management solution, so that you can customize it to improve the performance of Interact.

Interact | Cache Managers | EHCache | Parameter Data

The configuration properties in this category control how the EHCache cache management system works to improve the performance of Interact.

cacheType

Description

You can configure the Interact runtime servers in a server group to use a multicast address for sharing cache data. This is referred to as a *distributed cache*. The cacheType parameter specifies whether you are using the built-in EHCache caching mechanism in **local** (stand-alone) mode or **distributed** (as with a runtime server group).

Note:

If you select **Distributed** for the cacheType, all of the servers sharing the cache must be part of the same, single server group. You must also enable multicast to work between all members of a server group.

Default value

Local

Valid Values

Local | Distributed

multicastIPAddress

Description

If you specify that the **cacheType** parameter is "distributed," you are configuring the cache to operate via multicast between all members of an Interact runtime server group. The multicastIPAddress value is the IP address that all the Interact servers for the server group use for listening.

The IP address must be unique across your server groups.

Default value

230.0.0.1

multicastPort

Description

If you specify that the **cacheType** parameter is "distributed," the **multicastPort** parameter indicates the port that all of the Interact servers for the server group use for listening.

Default value

6363

overflowToDisk

Description

The EHCache cache manager manages the session information using available memory. For environments where the session size is large due to a large profile, the number of sessions to be supported in memory may not be large enough to support the customer scenario. For situations where this is the case, EHCache has an optional feature to allow cache information greater than the amount that can be kept in memory to be written temporarily to the hard drive instead.

If you set the **overflowToDisk** property to "yes," each Java virtual machine (JVM) can handle more concurrent sessions than the memory alone would have allowed.

Default value

No

Valid Values

No | Yes

diskStore

Description

When the configuration property **overflowToDisk** is set to `Yes`, this configuration property specifies the disk directory that will hold the cache entries that are overflowed from memory. If this configuration property does not exist or its value is not valid, the disk directory is automatically created in the operating system's default temporary directory.

Default value

None

Valid Values

A directory to which the web application hosting Interact run time has write privileges.

(Parameter)

Description

A template that you can use to create a custom parameter to be used with the cache manager. You can set up any parameter name, and the value it must have.

To create a custom parameter, click ***(Parameter)*** and complete the name and the value you want to assign to that parameter. When you click **Save Changes**, the parameter you have created is added to the list in the Parameter Data category.

Default value

None

Interact | cacheManagement | Cache Managers | Extreme Scale

The Extreme Scale category specifies the parameters for the adapter to use the WebSphere eXtreme Scale cache management solution, so that you can customize it to improve the performance of Interact.

ClassName

Description

The fully-qualified name of the class that connects Interact to the WebSphere eXtreme Scale server. It must be `com.unicacorp.interact.cache.extremescale.ExtremeScaleCacheManager`.

Default value

`com.unicacorp.interact.cache.extremescale.ExtremeScaleCacheManager`

ClassPath

Description

The URI of the location of the file `interact_wxs_adapter.jar`, such as `file:///IBM/IMS/Interact/lib/interact_wxs_adapter.jar` or `file:///C:/IBM/IMS/Interact/lib/interact_wxs_adapter.jar`. However, if this jar file is already included in the class path of the hosting application server, this field should be left blank.

Default value

Blank

Interact | Cache Managers | Extreme Scale | Parameter Data

The configuration properties in this category control the WebSphere eXtreme Scale adapter that is optionally included with your Interact installation. These settings must be configured for each Interact run time server that is acting as a client to the eXtreme Scale server grid.

catalogPropertyFile

Description

The URI of the location of the property file used to start the WebSphere eXtreme Scale catalog server. If the Extreme Scale Adapter is used to start the catalog server, this property must be set. Otherwise, it will not be used.

Default value

```
file:///C:/depot/Interact/dev/main/extremescale/config/
catalogServer.props
```

containerPropertyFile

Description

The URI of the location of the property file used to start the WebSphere eXtreme Scale container instances. If the included server component is used to start the WebSphere eXtreme Scale container servers, this property must be set. Otherwise, it is not used.

Default value

```
file:///C:/depot/Interact/dev/main/extremescale/config/
containerServer.props
```

deploymentPolicyFile

Description

The URI of the location of the deployment policy file used to start the WebSphere eXtreme Scale catalog server. If the included server component is used to start the WebSphere eXtreme Scale catalog server, this property must be set. Otherwise, it is not used.

Default value

```
file:///C:/depot/Interact/dev/main/extremescale/config/
deployment.xml
```

objectGridConfigFile

Description

The URI of the location of the object grid configuration file used to start the WebSphere eXtreme Scale catalog server and also the near-cache component that runs together with the Interact run time server in the same Java Virtual Machine (JVM).

Default value

```
file:///C:/depot/Interact/dev/main/extremescale/config/
objectgrid.xml
```

gridName

Description

The name of the WebSphere eXtreme Scale grid that holds all Interact caches.

Default value

InteractGrid

catalogURLs

Description

A URL containing the host name or IP address and the port on which the WebSphere eXtreme Scale catalog server is listening for connections.

Default value

None

(Parameter)

Description

A template that you can use to create a custom parameter to be used with the cache manager. You can set up any parameter name, and the value it must have.

To create a custom parameter, click ***(Parameter)*** and complete the name and the value you want to assign to that parameter. When you click **Save Changes**, the parameter you have created is added to the list in the Parameter Data category.

Default value

None

Interact | caches

Use this set of configuration properties to specify which supported cache manager you want to use to improve the performance of Interact, such as Ehcache or WebSphere eXtreme Scale caching, and to configure specific cache properties for the runtime server you are configuring.

This includes the caches for storing session data, event pattern states, and segmentation results. By adjusting those settings, you can specify which cache solution to use for each type of caching, and you can specify individual settings to control how the cache works.

Interact | cacheManagement | caches | InteractCache

The InteractCache category configures the caching for all session objects, including the profile data, segmentation results, most recently delivered treatments, parameters passed through API methods, and other objects used by the Interact run time.

The InteractCache category is required for Interact to work properly.

The InteractCache category can also be configured through an external EHCache configuration for settings that are not supported in **Interact | cacheManagement | Caches**. If you use EHCache, you must ensure that InteractCache is configured properly.

CacheManagerName

Description

The name of the cache manager that handles the Interact cache. The value you enter here must be one of the cache managers defined in the **Interact | cacheManagement | Cache Managers** configuration properties, such as `EHCache` or `Extreme Scale`.

Default value

EHCache

Valid Values

Any cache manager defined in the **Interact | cacheManagement | Cache Managers** configuration property.

maxEntriesInCache

Description

The maximum number of session data objects to store in this cache. When the maximum number of session data objects has been reached, and data for an additional session need to be stored, the least-recently used object is deleted.

Default value

100000

Valid Values

Integer greater than 0.

timeoutInSecs

Description

The time in seconds that have elapsed since a session data object has been used or updated that are used to determine when the object is removed from the cache.

Note: If you upgraded from a version prior to 9.1, then you will need to reconfigure `timeoutInSecs` property because the property moved.

Default value

300

Valid Values

Integer greater than 0.

Interact | Caches | Interact Cache | Parameter Data

The configuration properties in this category control the Interact Cache that is automatically used by your Interact installation. These settings must be configured individually for each Interact run time server.

asyncIntervalMillis

Description

The time in millisecond that the cache manager EHCache should wait before it replicates any changes to other Interact run time instances. If the value is not positive, those changes will be replicated synchronously.

This configuration property is not created by default. If you create this property, it is used only when EHCache is the cache manager, and when the ehCache **cacheType** property is set to `distributed`.

Default value

None.

(Parameter)

Description

A template that you can use to create a custom parameter to be used with the Intearct Cache. You can set up any parameter name, and the value it must have.

To create a custom parameter, click ***(Parameter)*** and complete the name and the value you want to assign to that parameter. When you click **Save Changes**, the parameter you have created is added to the list in the Parameter Data category.

Default value

None

Interact | cacheManagement | caches | PatternStateCache

The PatternStateCache category is used to host the states of event patterns and real time offer suppression rules. By default, this cache is configured as a read-through and write-through cache, so that Interact attempts to use the cache first event pattern and offer suppression data. If the requested entry does not exist in the cache, the cache implementation loads it from the data source, through either the JNDI configuration or directly using a JDBC connection.

To use a JNDI connection, Interact connects to an existing data source provider that has been defined through the specified server using the JNDI name, URL, and so on. For a JDBC connection, you must provide a set of JDBC settings that include the JDBC driver class name, database URL, and authentication information.

Note that if you define multiple JNDI and JDBC sources, the first enabled JNDI source is used, and if there is no enabled JNDI sources, the first enabled JDBC source is used.

The PatternStateCache category is required for Interact to work properly.

The PatternStateCache category can also be configured through an external EHCache configuration for settings that are not supported in **Interact | cacheManagement | Caches**. If you use EHCache, you must ensure that PatternStateCache is configured properly.

CacheManagerName

Description

The name of the cache manager that handles the Interact pattern state cache. The value you enter here must be one of the cache managers defined in the **Interact | cacheManagement | Cache Managers** configuration properties, such as `EHCache` or `Extreme Scale`.

Default value

EHCache

Valid Values

Any cache manager defined in the **Interact | cacheManagement | Cache Managers** configuration property.

maxEntriesInCache

Description

The maximum number of event pattern states to store in this cache. When the maximum number of event pattern states has been reached, and data for an additional event pattern state need to be stored, the least-recently used object is deleted.

Default value

100000

Valid Values

Integer greater than 0.

timeoutInSecs

Description

Specifies the amount of time, in seconds, for an event pattern state object to time out in the event pattern state cache. When such a state object has been idling in the cache for `timeoutInSecs` number of seconds, it may be ejected from the cache based on the least-recently-used rule. Note that the value of this property should be larger than that defined in the `sessionTimeoutInSecs` property.

Note: If you upgraded from a version prior to 9.1, then you will need to reconfigure `timeoutInSecs` property because the property moved.

Default value

300

Valid Values

Integer greater than 0.

Interact | Caches | PatternStateCache | Parameter Data:

The configuration properties in this category control the Pattern State Cache used to host the states of event patterns and real time offer suppression rules.

(Parameter)

Description

A template that you can use to create a custom parameter to be used with the Pattern State Cache. You can set up any parameter name, and the value it must have.

To create a custom parameter, click ***(Parameter)*** and complete the name and the value you want to assign to that parameter. When you click **Save Changes**, the parameter you have created is added to the list in the Parameter Data category.

Default value

None

Interact | cacheManagement | caches | PatternStateCache | loaderWriter:

The **loaderWriter** category contains the configuration of the loader that interacts with external repositories for the retrieval and persistence of event patterns.

className

Description

The fully-qualified class name for this loader. This class must comply with the chosen cache manager's requirement.

Default value

`com.unicacorp.interact.cache.ehcache.loaderwriter.PatternStateEHCacheLoaderWriter`

Valid Values

A fully-qualified class name.

classPath

Description

The path to the loader's class file. If you leave this value blank or the entry is invalid, the class path used for running Interact is used.

Default value

None

Valid Values

A valid class path.

writeMode

Description

Specifies the mode for the writer to persist the new or updated event pattern states in the cache. Valid options are:

- WRITE_THROUGH. Every time there is a new entry or an existing entry is updated, that entry is written into the repositories immediately.
- WRITE_BEHIND. The cache manager waits for some time to collect a number of changes, and then persists them into the repositories in a batch.

Default value

WRITE_THROUGH

Valid Values

WRITE_THROUGH or WRITE_BEHIND.

batchSize

Description

The maximum number of event pattern state objects the writer will persist in a batch. This property is used only when **writeMode** is set to `WRITE_BEHIND`.

Default value

100

Valid Values

Integer value.

maxDelayInSecs

Description

The maximum time in seconds that the cache manager waits before an event pattern state object is persisted. This property is used only when **writeMode** is set to `WRITE_BEHIND`.

Default value

5

Valid Values

Integer value.

Interact | Caches | PatternStateCache | loaderWriter | Parameter Data:

The configuration properties in this category control the Pattern State Cache loader.

(Parameter)

Description

A template that you can use to create a custom parameter to be used with the Pattern State Cache loader. You can set up any parameter name, and the value it must have.

To create a custom parameter, click ***(Parameter)*** and complete the name and the value you want to assign to that parameter. When you click **Save Changes**, the parameter you have created is added to the list in the Parameter Data category.

Default value

None

Interact | cacheManagement | caches | PatternStateCache | loaderWriter | jndiSettings:

The **jndiSettings** category contains the configuration for the JNDI data source the loader will use to communicate with the backing database. To create a new set of JNDI settings, expand the **jdniSettings** category and click the ***(jndiSetting)*** property.

(jndiSettings)

Note: When the WebSphere Application Server is used, the loaderWriter is not get connected with the **jndiSettings**.

Description

When you click this category, a form appears. To define a JNDI data source, complete the following values:

- **New category name** is the name you want to use to identify this JNDI connection.
- **enabled** lets you indicate whether you want this JNDI connection to be available for use or not. Set this to `True` for new connections.
- **jdniName** is the JNDI name that has already been defined in the data source when it was set up.

- **providerUrl** is the URL to find this JNDI data source. If you leave this field blank, the URL of the web application that hosts the Interact run time is used.
- **Initial context factory** is the fully qualified class name of the initial context factory class for connecting to the JNDI provider. If the web application hosting the Interact run time is used for the **providerUrl**, leave this field blank.

Default value

None.

Interact | cacheManagement | caches | PatternStateCache | loaderWriter | jdbcSettings:

The **jdbcSettings** category contains the configuration for the JDBC connections the loader will use to communicate with the backing database. To create a new set of JDBC settings, expand the **jdbcSettings** category and click the ***(jdbcSetting)*** property.

(jdbcSettings)

Description

When you click this category, a form appears. To define a JDBC data source, complete the following values:

- **New category name** is the name you want to use to identify this JDBC connection.
- **enabled** lets you indicate whether you want this JDBC connection to be available for use or not. Set this to `True` for new connections.
- **driverClassName** is the fully-qualified class name of the JDBC driver. This class must exist in the class path configured for starting the hosting cache server.
- **databaseUrl** is the URL to find this JDBC data source.
- **asmUser** is the name of the IBM Marketing Software user that has been configured with the credentials for connecting to the database in this JDBC connection.
- **asmDataSource** the name of IBM Marketing Software data source that has been configured with the credentials for connecting to the database in this JDBC connection.
- **maxConnection** is the maximum number of concurrent connections that are allowed to be made the database in this JDBC connection.

Default value

None.

Interact | triggeredMessage

The configuration properties in this category define settings for all triggered messages and offer channel delivery.

backendProcessIntervalMin

Description

This property defines the time period in minutes that the backend thread loads and processes delayed offer deliveries. This value must be an integer. If the value is zero or negative, the backend process is disabled.

Valid Values

A positive integer

autoLogContactAfterDelivery

Description

If this property is set to true, a contact event is automatically posted as soon as this offer is dispatched or this offer is queued for delayed delivery If this property is set to false, no contact event is automatically posted for the outbound offers. This is the default behavior.

Note:

- If you want to capture additional attributes in the contact history when the outbound message is triggered, you can add the additional custom attributes as columns in the contact history. While posting an event, that would trigger the outbound triggered message, you can pass values for the attributes in the `postEvent` method as the name value parameters
- To parametrize an offer to an outbound channel, you could assign offers in the associated strategy, deploy the channel, personalize the offer, and in the triggered message choose **Automatically select next best offer**.

Valid Values

True | False

waitForFlowchart

Description

This property determines if the flowchart should wait for the currently running segmentation to finish, and the behavior if that wait times out.

DoNotWait: The processing of a triggered message starts regardless if segmentation is currently running or not. However, if segments are used in the eligibility rule and/or NextBestOffer is selected as the offer selection method, the TM execution still waits.

OptionalWait : The processing of a triggered message waits until the currently running segmentation finishes or times out. If the wait times out, a warning is logged, and the processing of this triggered message continues. This is the default.

MandatoryWait: The processing of a triggered message waits until the currently running segmentation finishes or times out. If the wait times out, an error is logged, and the processing of this triggered message aborts.

Valid Values

DoNotWait | OptionalWait | MandatoryWait

Interact | triggeredMessage | offerSelection

The configuration properties in this category define settings for offer selection in triggered messages.

maxCandidateOffers

Description

This property defines the maximum number of eligible offers that the engine returns to get the best offer for delivery. There is a chance that none of those returned eligible offers can be sent based on the selected channel. The more candidate offers there are, the less this case happens. However, more candidate offers can increase processing time.

Valid Values

A positive integer

defaultCellCode

Description

If the delivered offer is the result of evaluating a strategic rule or a table driven record, there is a target cell associated to it, and the information of this cell is used in all the related logging. However, if a list of specific offers are used as the input to the offer selection, no target cell is available. In this case, the value of this configuration setting is used. You must make sure this target cell and its campaign are included in the deployment. The easiest method to achieve this is to add the cell into a deployed strategy.

Interact | triggeredMessage | dispatchers

The configuration properties in this category define settings for all dispatchers in triggered messages.

dispatchingThreads

Description

This property defines the number of threads the engine uses to asynchronously call the dispatchers. If the value is 0 or a negative number, the invocation of dispatchers is synchronous. The default value is 0.

Valid Values

An integer

Interact | triggeredMessage | dispatchers | <dispatcherName>

The configuration properties in this category define settings for a specific dispatcher in triggered messages.

category name

Description

This property defines the name of this dispatcher. The name must be unique among all dispatchers.

type

Description

This property defines the disptacher type.

Valid Values

InMemoryQueue | JMSQueue | Custom

Note: If you use JMSQueue or Custom, to integrate Interact with IBM MQ, Interact runtime must be on appserver with JDK 1.7. For WebSphere and WebLogic, it is recommended to use the latest supplied JDK fix pack version.

JMSQueue only supports WebLogic. You cannot use JMSQueue if you use WebSphere Application Server.

className

Description

This property defines the fully qualified class name of this dispatcher implementation. If the type is InMemoryQueue the value should be empty. If the type is custom, this setting must have the value `com.unicacorp.interact.eventhandler.triggeredmessage.dispatchers.IBMMQDispatcher`.

classPath

Description

This property defines the URL to the JAR file that includes the implementation of this dispatcher.

If the type is custom, this setting much have the value `file://<Interact_HOME>/lib/interact_ibmmqdispatcher.jar;file://<Interact_HOME>/lib/com.ibm.mq.allclient.jar;file://<Interact_HOME>/lib/jms.jar`

Interact | triggeredMessage | dispatchers | <dispatcherName> | Parameter Data

The configuration properties in this category define parameters for a specific dispatcher in triggered messages.

You can choose between three types of dispatchers. InMemoryQueue is the internal dispatcher for Interact. Custom is used for IBM MQ. JMSQueue is used to connect to a JMS provider via JNDI.

cateogry name

Description

This property defines the name of this parameter. The name must be unique among all parameters for that dispatcher.

value

Description

This property defines the parameters, in the format of name value pairs, needed by this dispatcher.

Note: All parameters for trigger messages are case sensitive and should be entered as shown here.

If the type is InMemoryQueue, the following parameter is supported.

- queueCapacity: Optional. The maximum offers that can be waiting in the queue to be dispatched. When specified, this property must be a positive integer. If not specified or invalid, the default value (1000) is used.

If the type is Custom, the following parameters are supported.

- providerUrl: <hostname>:port (case sensitive)
- queueManager: The name of the queue manager that was created on the IBM MQ server.
- messageQueueName: The name of the message queue that was created on the IBM MQ server.
- enableConsumer: This property must be set to true.
- asmUserforMQAuth: The user name for logging into the server. It is required when the server enforces authentication. Otherwise, it should not be specified.
- authDS: The password associated with the user name for logging into the server. It is required when the server enforces authentication. Otherwise, it should not be specified.

If the type is JMSQueue, the following parameter is supported.

- providerUrl: The URL to the JNDI provider (case sensitive).
- connectionFactoryJNDI: The JNDI name of the JMS connection factory.
- messageQueueJNDI: The JNDI name of the JMS queue where the triggered messages are sent to and retrieved from.
- enableConsumer: Whether a consumer of those triggered messages should be started in Interact. This property must be set to true. If not specified, the default value (false) is used.
- initialContextFactory: The fully qualified name of the JNDI initial context factory class. IF you use WebLogic, the value of this parameter should be `weblogic.jndi.WLInitialContextFactory`.

Interact | triggeredMessage | gateways | <gatewayName>

The configuration properties in this category define settings for a specific gateway in triggered messages.

Interact does not support multiple instances of the same gateway. All of the gateway configuration files should be accessible from every Interact Runtime node. In the case of a distributed setup, ensure that the gateway files are kept at a shared location.

category name

Description

This proerpty defines the name of this gateway. It must be unique among all gateways.

className

Description

This property defines the fully qualified class name of this gateway implementation.

classPath

Description

This property defines the URI of the JAR file that includes the implementation of this gateway. If left empty, the class path of the hosting Interact application is used.

For example in a windows system, if the gateway JAR file is available in the directory, `C:\IBM\EMM\EmailGateway\IBM_Interact_OMO_OutboundGateway_Silverpop_1.0\lib\OMO_OutboundGateway_Silverpop.jar`, the classPath should be `file:///C:/IBM/EMM/EmailGateway/IBM_Interact_OMO_OutboundGateway_Silverpop_1.0/lib/OMO_OutboundGateway_Silverpop.jar`. In a unix system, if the gateway jar file is available in the directory, `/opt/IBM/EMM/EmailGateway/IBM_Interact_OMO_OutboundGateway_Silverpop_1.0/lib/OMO_OutboundGateway_Silverpop.jar`, the classpath should be `file:///opt/IBM/EMM/EmailGateway/IBM_Interact_OMO_OutboundGateway_Silverpop_1.0/lib/OMO_OutboundGateway_Silverpop.jar`.

Interact | triggeredMessage | gateways | <gatewayName> | Parameter Data

The configuration properties in this category define parameters for a specific gateway in triggered messages.

cateogry name

Description

This property defines the name of this parameter. The name must be unique among all parameters for that gateway.

value

Description

This property defines the parameters, in the format of name value pairs, needed by this gateway. For all gateways, the following parameters are supported.

Note: All parameters for trigger messages are case sensitive and should be entered as shown here.

- validationTimeoutMillis: The duration in milliseconds that the validation of an offer through this gateway timeouts. The default value is 500.
- deliveryTimeoutMillis: The duration in milliseconds that the delivery of an offer using this gateway timeouts. The default value is 1000.

Interact | triggeredMessage | channels

The configuration properties in this category define settings for all channels in triggered messages.

type

Description

This property defines the root node for settings related to a specific gateway. Default uses the built in channel selector, which is based on the list of channels defined on in the triggered messages UI. If default is selected, className and classPath values should be left blank. Custom uses the customer implementation of IChannelSelector.

Valid Values

Default | Custom

className

Description

This property defines the fully qualified class name of the customer implementation of channel selector. This setting is required if the type is Custom.

classPath

Description

This property defines the URL to the JAR file that includes the implementation of the customer implementation of channel selector. If left empty, the class path of the hosting Interact application is used.

Interact | triggeredMessage | channels | Parameter Data

The configuration properties in this category define parameters for a specific channel in triggered messages.

category name

Description

This property defines the name of this parameter. The name must be unique among all parameters for that channel.

value

Description

This property defines the parameters, in the format of name value pairs, needed by the channel selector.

If you use **Customer Preferred Channels** for your channel, you must create

Interact | triggeredMessage | channels | <channelName>

The configuration properties in this category define settings for a specific channel in triggered messages.

category name

Description

This property defines the name of the channel through which offers are sent. It should match those defined in the design time under **Campaign | partitions | <partition[N]> | Interact | outboundChannels**.

Interact | triggeredMessage | channels | <channelName> | <handlerName>

The configuration properties in this category define settings for a specific handler in triggered messages that is used to sent offers.

category name

Description

This property defines the name of the handler which the channel will use to send offers.

dispatcher

Description

This property defines the name of the dispatcher through which this handler uses send offers to the gateway. It must be one of those defined under **interact | triggeredMessage | dispatchers**.

gateway

Description

This property defines the name of the gateway to which this handler send offers ultimately. It must be one of those defined under **interact | triggeredMessage | gateways**.

mode

Description

This property defines the usage mode of this handler. If Failover is selected, this handler is used only when all the handlers with higher priorities defined within this channel failed to send offers. If Addon is selected, this handler is used no matter if other handlers have successfully sent offers.

priority

Description

This property defines the priority of this handler. The engine first tries to use the handler with the highest priority for sending offers.

Valid Values

Any integer

Default

100

Interact | activityOrchestrator

The activity orchestrator category specifies the receivers and gateways for your Interact inbound gateway activity.

Use the **Interact | activityOrchestrator | receivers** configuration properties to configure your Interact receivers. Use the **Interact | activityOrchestrator | gateways** configuration properties to configure your gateways to use in Interact.

Interact | activityOrchestrator | receivers

The activity orchestrator receivers category specifies the event receivers for your Interact inbound gateway activity.

Category name

Description

The name of your receiver.

Type

Description

The type of receiver. You can choose between IBM MQ and Custom. Custom requires you to use an implementation of the iReceiver.

Enabled

Description

Select True to enable the receiver or false to disable the receiver.

className

Description

This property defines the fully qualified class name of this receiver implementation. It is used only when the type is Custom.

classPath

Description

This property defines the URI to the JAR file that includes the implementation of this receiver. If left empty, the class path of the hosting Interact application is used. It is used only when the type is Custom.

Interact | activityOrchestrator | receivers | Parameter Data

You can add receiver parameters, such as queueManager and messageQueueName to define your receiver queue.

Interact | activityOrchestrator | gateways

The activity orchestrator gateway category specifies the gateways for your Interact inbound gateway activity.

Category name

Description

The name of your gateway.

className

Description

This property defines the fully qualified class name of this gateway implementation.

classPath

Description

This property defines the URI to the JAR file that includes the implementation of this gateway. If left empty, the class path of the hosting Interact application is used. It is used only when the type is Custom.

Interact | activityOrchestrator | gateways | Parameter Data

You can add gateway parameters for your gateway configuration files, such as OMO-conf_inbound_UBX_interactEventNameMapping and OMO-conf_inbound_UBX_interactEventPayloadMapping.

Interact | ETL | patternStateETL

The configuration properties in this category define the settings for the ETL process.

New category name

Description

Provide a name that uniquely identifies this configuration. Note that you must provide this exact name when you run the stand-alone ETL process. For convenience in specifying this name on the command line, you may want to avoid a name containing spaces or punctuation, such as `ETLProfile1`.

runOnceADay

Description

Determines whether the stand-alone ETL process in this configuration should run once each day. Valid answers are **Yes** or **No**. If you answer **No** here, the **processSleepIntervalInMinutes** determines the run schedule for the process.

preferredStartTime

Description

The preferred time at which the stand-alone ETL process should start. Specify the time in the format HH:MM:SS AM/PM, as in `01:00:00 AM`.

preferredEndTime

Description

The preferred time at which the stand-alone ETL process should stop. Specify the time in the format HH:MM:SS AM/PM, as in `08:00:00 AM`.

processSleepIntervalInMinutes

Description

If you have not configured the stand-alone ETL process to run once a day (as specified in the **runOnceADay** property), this property specifies the interval between ETL process runs. For example, if you specify 15 here, the stand-alone ETL process will wait for 15 minutes after it stops running before starting the process again.

maxJDBCInsertBatchSize

Description

The maximum number of records of a JDBC batch before committing the query. By default, this is set to 5000. Note that this is not the maximum number of records that the ETL processes in one iteration. During each iteration, the ETL processes all available records from the UACI_EVENTPATTERNSTATE table. However, all those records are broken into **maxJDBCInsertSize** chunks.

maxJDBCFetchBatchSize

Description

The maximum number of records of a JDBC batch to fetch from the staging database.

You may need to increase this value to tune the performance of the ETL.

communicationPort

Description

The network port on which the standalone ETL process listens for a stop request. Under normal circumstances, there should be no reason to change this from the default value.

queueLength

Description

A value used for performance tuning. Collections of pattern state data are fetched and transformed into objects that are added to a queue to be processed and written to the database. This property controls the size of the queue.

completionNotificationScript

Description

Specifies the absolute path to a script to run when the ETL process is completed. If you specify a script, three arguments are passed to the completion notification script: start time, end time, and total number of event pattern records processed. The start time and end time are numeric values representing number of milliseconds elapsed since 1970.

Interact | ETL | patternStateETL | <patternStateETLName> | RuntimeDS

The configuration properties in this category define the settings for the ETL Runtime DS.

type

Description

A list of the supported database types for the data source you are defining.

dsname

Description

The JNDI name of the data source. This name must also be used in the user's data source configuration to ensure that the user has access to the target and runtime data sources.

driver

Description

The name of the JDBC driver to us, such as any of the following:

Oracle: `oracle.jdbc.OracleDriver`

Microsoft SQL Server: `com.microsoft.sqlserver.jdbc.SQLServerDriver`

IBM DB2: `com.ibm.db2.jcc.DB2Driver`

serverURL

Description

The data source URL, such as any of the following:

Oracle: `jdbc:oracle:thin:@ <your_db_host>:<your_db_port>:<your_db_service_name>`

Microsoft SQL Server: `jdbc:sqlserver:// <your_db_host>:<your_db_port> ;databaseName= <your_db_name>`

IBM DB2: `jdbc:db2:// <your_db_host>:<your_db_port>/<your_db_name>`

connectionpoolSize

Description

A value indicating the size of the connection pool, provided for performance tuning. Pattern state data is read and transformed concurrently depending upon the available database connections. Increasing the connection pool size allows for more concurrent database connections, subject to limitations of memory and database read/write capabilities. For example, if this value is set to 4, four jobs will run concurrently. If you have a large amount of data, you might need to increase this value to a number such as 10 or 20, as long as sufficient memory and database performance is available.

schema

Description

The name of the database schema to which this configuration is connecting.

connectionRetryPeriod

Description

The `ConnectionRetryPeriod` property specifies the amount of time in seconds Interact automatically retries the database connection request on failure. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact retries indefinitely; if the value is set to -1, no retry is attempted.

connectionRetryDelay

Description

The `ConnectionRetryDelay` property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure. If the value is set to -1, no retry is attempted.

Interact | ETL | patternStateETL | <patternStateETLName> | TargetDS

The configuration properties in this category define the settings for the ETL Target DS.

type

Description

A list of the supported database types for the data source you are defining.

dsname

Description

The JNDI name of the data source. This name must also be used in the user's data source configuration to ensure that the user has access to the target and runtime data sources.

driver

Description

The name of the JDBC driver to us, such as any of the following:

Oracle: `oracle.jdbc.OracleDriver`

Microsoft SQL Server: `com.microsoft.sqlserver.jdbc.SQLServerDriver`

IBM DB2: `com.ibm.db2.jcc.DB2Driver`

serverURL

Description

The data source URL, such as any of the following:

Oracle: `jdbc:oracle:thin:@ <your_db_host>:<your_db_port>:<your_db_service_name>`

Microsoft SQL Server: `jdbc:sqlserver:// <your_db_host>:<your_db_port> ;databaseName= <your_db_name>`

IBM DB2: `jdbc:db2:// <your_db_host>:<your_db_port>/<your_db_name>`

connectionpoolSize

Description

A value indicating the size of the connection pool, provided for performance tuning. Pattern state data is read and transformed concurrently depending upon the available database connections. Increasing the connection pool size allows for more concurrent database connections, subject to limitations of memory and database read/write capabilities. For example, if this value is set to 4, four jobs will run concurrently. If you have a large amount of data, you might need to increase this value to a number such as 10 or 20, as long as sufficient memory and database performance is available.

schema

Description

The name of the database schema to which this configuration is connecting.

connectionRetryPeriod

Description

The ConnectionRetryPeriod property specifies the amount of time in seconds Interact automatically retries the database connection request on failure. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact retries indefinitely; if the value is set to -1, no retry is attempted.

connectionRetryDelay

Description

The ConnectionRetryDelay property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure. If the value is set to -1, no retry is attempted.

Interact | ETL | patternStateETL | <patternStateETLName> | Report

The configuration properties in this category define the settings for the ETL report aggregation process.

enable

Description

Enable or disable the report integration with ETL. This property is set to disable by default.

If set to disable, this property disables updates on table UARI_DELTA_PATTERNS Table . It does not disable reporting completely .

Note: To disable the report integration with ETL, you must also alter the trigger TR_AGGREGATE_DELTA_PATTERNS to disable on UACI_ETLPATTERNSTATERUN staging table.

retryAttemptsIfAggregationRunning

Description

The number of times the ETL attempts to check whether the report aggregation is completed if the lock flag is set. This property is set to 3 by default.

sleepBeforeRetryDurationInMinutes

Description

Sleep time in minutes between consecutive attempts. This property is set to 5 minutes by default.

aggregationRunningCheckSql

Description

This property lets you define a custom SQL, which can be run to see whether the report aggregation lock flag is set. By default this property is empty.

When this property is not set, the ETL runs the following SQL to get the lock flag.

```
select count(1) AS ACTIVERUNS from uari_pattern_lock where islock='Y'
=> If ACTIVERUNS is > 0, lock is set
```

aggregationRunningCheck

Description

Enable or disable the check if the report aggregation is running before the ETL run is performed. This property is set to enable by default.

Chapter 14. Interact Simulator

This section describes all the configuration properties for the Interact simulator.

Interact | simulator

The configuration category defines the parameters to be defined to run the coverage analysis scenario of Simulator module..

numberOfThreads

Description

The number of threads used to run the simulation

Default Value

1

maxOffersToInclude

Description

The maximum number of offers returned in each getOffers call for each audience id in the coverage analysis scenario.

Default Value

10

insertBatchSize

Description

Define the size of each batch for persisting the resulting records.

Default Value

1000

Interact | simulator|scenarioDataSource

These configurations are required to run Simulator Coverage Analysis scenario

jndiName

Description

Use this jndiName property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (Websphere or WebLogic) for the Interact Design Time tables.

Default Value

No default value defined.

Schema

Description

The name of the schema containing the tables for the Interact design time data source

module. Interact inserts the value of this property before all table names,

for example, UACI_IntChannel becomes schema.UACI_IntChannel.

You have to define a schema. If you do not define a schema, Interact assumes that the owner of the tables is the same as the schema. It is required to specify schema name to run coverage scenario successfully.

Default Value

No Default value defined.

type

Description

The database type for the data source used by the Interact Design time tables accessed

by the Interact Simulator.

Default Value

sqlserver

Valid Value

sqlserver | Db2 | Oracle

connectionRetryPeriod

Description

The ConnectionRetryPeriod property specifies the amount of time in seconds Interact automatically retries the database connection request on failure for the learning tables. Interact automatically tries to reconnect to the database for this length of time before reporting a database error or failure. If the value is set to 0, Interact will retry indefinitely; if the value is set to -1, no retry will be attempted.

Default Value

-1

connectionRetryDelay

Description

The ConnectionRetryDelay property specifies the amount of time in seconds Interact waits before it tries to reconnect to the database after a failure for the learning tables. If the value is set to -1, no retry will be attempted.

Default Value

-1

Error Handling for Simulator

This section lists the status codes the application writes into the table UACI_SimulationHistory which is present in the Interact Design time database.

In case of an error the application will show the scenario failed message on the Simulator run page. The detailed status code can be found in the database table UACI_SimulationHistory.

The following are the list of possible status codes that a scenario run history could have:

// status code 0-99 are for information

Status	**Code**	**Severity level**	**Http Status**	**Possible UI message**
SUCCESS	0	*INFO*	*OK*	Running simulation succeeded
RUNNING	1	*INFO*	*OK*	Running
CANCELING	2	*INFO*	*OK*	Cancelling
CANCELED	3	*INFO*	*OK*	Cancelled
EXPORTING_TO_CSV	4	*INFO*	*OK*	Exporting to CSV
EXPORTED_TO_CSV	5	*INFO*	*OK*	Exported to CSV

// status code 101-999 are for errors

Status	**Code**	**Severity level**	**Http Status**	**Possible UI message**
NOT_ENABLED	101	*WARN*	SERVICE_UNAVAILABLE	Simulation is not enabled on this run time server
ERROR_RETRIEVE_SCENARIO	102	ERROR	INTERNAL_SERVER_ERROR	Error retrieving the scenario information for simulation
INVALID_SCENARIO	103	ERROR	BAD_REQUEST	Invalid scenario information of simulation
ERROR_CREATE_RESULT_TABLE	104	ERROR	INTERNAL_SERVER_ERROR	Error creating the table for storing results for simulation
ERROR_RETRIEVE_AUDIENCE	105	ERROR	INTERNAL_SERVER_ERROR	Error retrieving audience IDs for simulation

ERROR_CONNECT_DATABASE	106	ERROR	INTERNAL_SERVER_ERROR	Error connecting to {0} database for simulation
ERROR_PERSIST_RESULT	107	ERROR	INTERNAL_SERVER_ERROR	Error persisting results to database for simulation
SCENARIO_NOT_FOUND	108	ERROR	NOT_FOUND	Cannot find a scenario ready to run
GENERIC_ERROR	109	ERROR	INTERNAL_SERVER_ERROR	Server error running simulation
ERROR_UPDATE_RESULT	110	ERROR	INTERNAL_SERVER_ERROR	Error updating result for simulation
ERROR_INVALID_IC	111	ERROR	BAD_REQUEST	Interactive channel is not deployed

// // status code 1001 and above are for UI only, will not be stored in database

Status	**Code**	**Severity level**	**Http Status**	**Possible UI message**
SIMULATION_ALREADY_RUNNING	1001	WARN	PRECONDITION_FAILED	A simulation is already running for this scenario
SIMULATION_NOT_FOUND	1002	WARN	NO_CONTENT	No ongoing simulation found for this scenario
SIMULATION_RUNNING	1003	INFO	OK	Running
SIMULATION_NOT_RUNNING	1004	INFO	OK	Simulation not running

Chapter 15. Interact design environment configuration properties

This section describes all the configuration properties for Interact design environment.

Campaign | partitions | partition[n] | reports

The **Campaign | partitions | partition[n] | reports** property defines the different types of folders for reports.

offerAnalysisTabCachedFolder

Description

The `offerAnalysisTabCachedFolder` property specifies the location of the folder that contains the specification for bursted (expanded) offer reports listed on the Analysis tab when you reach it by clicking the Analysis link on the navigation pane. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign - Object Specific
Reports']/folder[@name='offer']/folder[@name='cached']
```

segmentAnalysisTabOnDemandFolder

Description

The `segmentAnalysisTabOnDemandFolder` property specifies the location of the folder that contains the segment reports listed on the Analysis tab of a segment. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign - Object Specific
Reports']/folder[@name='segment']/folder[@name='cached']
```

offerAnalysisTabOnDemandFolder

Description

The `offerAnalysisTabOnDemandFolder` property specifies the location of the folder that contains the offer reports listed on the Analysis tab of an offer. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign - Object Specific
Reports']/folder[@name='offer']
```

segmentAnalysisTabCachedFolder

Description

The `segmentAnalysisTabCachedFolder` property specifies the location of the folder that contains the specification for bursted (expanded) segment

reports listed on the Analysis tab when you reach it by clicking the Analysis link on the navigation pane. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign - Object Specific
Reports']/folder[@name='segment']
```

analysisSectionFolder

Description

The `analysisSectionFolder` property specifies the location of the root folder where report specifications are stored. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign']
```

campaignAnalysisTabOnDemandFolder

Description

The `campaignAnalysisTabOnDemandFolder` property specifies the location of the folder that contains the campaign reports listed on the Analysis tab of a campaign. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign - Object Specific
Reports']/folder[@name='campaign']
```

campaignAnalysisTabCachedFolder

Description

The `campaignAnalysisTabCachedFolder` property specifies the location of the folder that contains the specification for bursted (expanded) campaign reports listed on the Analysis tab when you reach it by clicking the Analysis link on the navigation pane. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign - Object Specific
Reports']/folder[@name='campaign']/folder[@name='cached']
```

campaignAnalysisTabEmessageOnDemandFolder

Description

The `campaignAnalysisTabEmessageOnDemandFolder` property specifies the location of the folder that contains the eMessage reports listed on the Analysis tab of a campaign. The path is specified by using the XPath notation.

Default value

```
/content/folder[@name='Affinium Campaign']/folder[@name='eMessage
Reports']
```

campaignAnalysisTabInteractOnDemandFolder

Description

Report server folder string for Interact reports.

Default value

/content/folder[@name='Affinium Campaign']/folder[@name='Interact Reports']

Availability

This property is applicable only if you install Interact.

interactiveChannelAnalysisTabOnDemandFolder

Description

Report server folder string for Interactive Channel analysis tab reports.

Default value

/content/folder[@name='Affinium Campaign - Object Specific Reports']/folder[@name='interactive channel']

Availability

This property is applicable only if you install Interact.

Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking

These configuration properties define settings for the Interact contact and response history module.

isEnabled

Description

If set to `yes`, enables the Interact contact and response history module which copies the Interact contact and response history from staging tables in the Interact runtime to the Campaign contact and response history tables. The property `interactInstalled` must also be set to `yes`.

Default value

no

Valid Values

yes | no

Availability

This property is applicable only if you have installed Interact.

runOnceADay

Description

Specifies whether to run the Contact and Response History ETL once a day. If you set this property to `Yes`, the ETL runs during the scheduled interval specified by `preferredStartTime` and `preferredEndTime`.

If ETL takes more than 24 hours to execute, and thus misses the start time for the next day, it will skip that day and run at the scheduled time the

following day. For example, if ETL is configured to run between 1AM to 3AM, and the process starts at 1AM on Monday and completes at 2AM on Tuesday, the next run, originally scheduled for 1AM on Tuesday, will be skipped, and the next ETL will start at 1AM on Wednesday.

ETL scheduling does not account for Daylight Savings Time changes. For example, if ETL scheduled to run between 1AM and 3AM, it could run at 12AM or 2AM when the DST change occurs.

Default value

No

Availability

This property is applicable only if you have installed Interact.

processSleepIntervalInMinutes

Description

The number of minutes the Interact contact and response history module waits between copying data from the Interact runtime staging tables to the Campaign contact and response history tables.

Default value

60

Valid Values

Any integer greater than zero.

Availability

This property is applicable only if you have installed Interact.

preferredStartTime

Description

The preferred time to start the daily ETL process. This property, when used in conjunction with the preferredEndTime property, sets up the preferred time interval during which you want the ETL to run. The ETL will start during the specified time interval and will process at most the number of records specified using `maxJDBCFetchBatchSize`. The format is HH:mm:ss AM or PM, using a 12-hour clock.

Default value

12:00:00 AM

Availability

This property is applicable only if you have installed Interact.

preferredEndTime

Description

The preferred time to complete the daily ETL process. This property, when used in conjunction with the preferredStartTime property, sets up the preferred time interval during which you want the ETL to run. The ETL will start during the specified time interval and will process at most the number of records specified using `maxJDBCFetchBatchSize`. The format is HH:mm:ss AM or PM, using a 12-hour clock.

Default value

2:00:00 AM

Availability

This property is applicable only if you have installed Interact.

purgeOrphanResponseThresholdInMinutes

Description

The number of minutes the Interact contact and response history module waits before purging responses with no corresponding contact. This prevents logging responses without logging contacts.

Default value

180

Valid Values

Any integer greater than zero.

Availability

This property is applicable only if you have installed Interact.

maxJDBCInsertBatchSize

Description

The maximum number of records of a JDBC batch before committing the query. This is not the max number of records that the Interact contact and response history module processes in one iteration. During each iteration, the Interact contact and response history module processes all available records from the staging tables. However, all those records are broken into `maxJDBCInsertSize` chunks.

Default value

1000

Valid Values

Any integer greater than zero.

Availability

This property is applicable only if you have installed Interact.

maxJDBCFetchBatchSize

Description

The maximum number of records of a JDBC batch to fetch from the staging database. You may need to increase this value to tune the performance of the contact and response history module.

For example, to process 2.5 million contact history records a day, you should set `maxJDBCFetchBatchSize` to a number greater than 2.5M so that all records for one day will be processed.

You could then set `maxJDBCFetchChunkSize` and `maxJDBCInsertBatchSize` to smaller values (in this example, perhaps to 50,000 and 10,000, respectively). Some records from the next day may be processed as well, but would then be retained until the next day.

Default value

1000

Valid Values

Any integer greater than zero

maxJDBCFetchChunkSize

Description

The maximum number of a JDBC chunk size of data read during ETL (extract, transform, load). In some cases, a chunk size greater than insert size can improve the speed of the ETL process.

Default value

1000

Valid Values

Any integer greater than zero

deleteProcessedRecords

Description

Specifies whether to retain contact history and response history records after they have been processed.

Default value

Yes

completionNotificationScript

Description

Specifies the absolute path to a script to run when the ETL is completed. If you specify a script, five arguments are passed to the completion notification script: start time, end time, total number of CH records processed, total number of RH records processed and status. The start time and end time are numeric values representing number of milliseconds elapsed since 1970. The status argument indicates whether the ETL job was a success or failure. 0 indicates a successful ETL job. 1 indicates a failure and that there are some errors in the ETL job.

Default value

None

fetchSize

Description

Allow you to set the JDBC fetchSize when retrieving records from staging tables.

On Oracle databases especially, adjust the setting to the number of records that the JDBC should retrieve with each network round trip. For large batches of 100K or more, try 10000. Be careful not to use too large a value here, because that will have an impact on memory usage and the gains will become negligible, if not detrimental.

Default value

None

daysBackInHistoryToLookupContact

Description

Limits the records that are searched during response history queries to those within the past specified number of days. For databases with a large number of response history records, this can reduce processing time on queries by limiting the search period to the number of days specified.

The default value of 0 indicates that all records are searched.

Default value

0 (zero)

Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking | runtimeDataSources | [runtimeDataSource]

These configuration properties define the data source for the Interact contact and response history module.

jndiName

Description

Use the `systemTablesDataSource` property to identify the Java Naming and Directory Interface (JNDI) data source that is defined in the application server (Websphere or WebLogic) for the Interact runtime tables.

The Interact runtime database is the database populated with the `aci_runtime` and `aci_populate_runtime` dll scripts and, for example, contains the following tables (among others): `UACI_CHOfferAttrib` and `UACI_DefaultedStat`.

Default value

No default value defined.

Availability

This property is applicable only if you have installed Interact.

databaseType

Description

Database type for the Interact runtime data source.

Default value

SQLServer

Valid Values

SQLServer | Oracle | DB2

Availability

This property is applicable only if you have installed Interact.

schemaName

Description

The name of the schema containing the contact and response history module staging tables. This should be the same as the runtime environment tables.

You do not have to define a schema.

Default value

No default value defined.

Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking | contactTypeMappings

These configuration properties define the contact type from campaign that maps to a 'contact' for reporting or learning purposes.

contacted

Description

The value assigned to the `ContactStatusID` column of the `UA_DtlContactHist` table in the Campaign system tables for an offer contact. The value must be a valid entry in the `UA_ContactStatus` table. See the *Campaign Administrator's Guide* for details on adding contact types.

Default value

2

Valid Values

An integer greater than zero.

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking | responseTypeMappings

These configuration properties define the responses for accept or reject for reporting and learning.

accept

Description

The value assigned to the `ResponseTypeID` column of the `UA_ResponseHistory` table in the Campaign system tables for an accepted offer. The value must be a valid entry in the `UA_UsrResponseType` table. You should assign the `CountsAsResponse` column the value `1`, a response.

See the *Campaign Administrator's Guide* for details on adding response types.

Default value

3

Valid Values

An integer greater than zero.

Availability

This property is applicable only if you have installed Interact.

reject

Description

The value assigned to the ResponseTypeID column of the UA_ResponseHistory table in the Campaign system tables for a rejected offer. The value must be a valid entry in the UA_UsrResponseType table. You should assign the CountsAsResponse column the value 2, a reject. See the *Campaign Administrator's Guide* for details on adding response types.

Default value

8

Valid Values

Any integer greater than zero.

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | report

These configuration properties define the report names when integrating with Cognos.

interactiveCellPerformanceByOfferReportName

Description

Name for Interactive Cell Performance by Offer report. This name must match the name of this report on the Cognos server.

Default value

Interactive Cell Performance by Offer

treatmentRuleInventoryReportName

Description

Name for Treatment Rule Inventory report. This name must match the name of this report on the Cognos server.

Default value

Channel Treatment Rule Inventory

deploymentHistoryReportName

Description

Name for Deployment History Report report. This name must match the name of this report on the Cognos server

Default value

Channel Deployment History

Campaign | partitions | partition[n] | Interact | learning

These configuration properties enable you to tune the built-in learning module.

confidenceLevel

Description

A percentage indicating how confident you want the learning utility to be before switching from exploration to exploitation. A value of 0 effectively shuts off exploration.

This property is applicable if the `Interact > offerserving > optimizationType` property for Interact runtime is set to `BuiltInLearning` only.

Default value

95

Valid Values

An integer between 0 and 95 divisible by 5 or 99.

validateonDeployment

Description

If set to `No`, Interact does not validate the learning module when you deploy. If set to `yes`, Interact validates the learning module when you deploy.

Default value

`No`

Valid Values

Yes | No

maxAttributeNames

Description

The maximum number of learning attributes the Interact learning utility monitors.

This property is applicable if the `Interact > offerserving > optimizationType` property for Interact runtime is set to `BuiltInLearning` only.

Default value

`10`

Valid Values

Any integer.

maxAttributeValues

Description

The maximum number of values the Interact learning module tracks for each learning attribute.

This property is applicable if the `Interact > offerserving > optimizationType` property for Interact runtime is set to `BuiltInLearning` only.

Default value

5

otherAttributeValue

Description

The default name for the attribute value used to represent all attribute values beyond the `maxAttributeValues`.

This property is applicable if the `Interact > offerserving > optimizationType` property for Interact runtime is set to `BuiltInLearning` only.

Default value

`Other`

Valid Values

A string or number.

Example

If maxAttributeValues is set to 3 and otherAttributeValue is set to other, the learning module tracks the first three values. All of the other values are assigned to the other category. For example, if you are tracking the visitor attribute hair color, and the first five visitors have the hair colors black, brown, blond, red, and gray, the learning utility tracks the hair colors black, brown, and blond. The colors red and gray are grouped under the `otherAttributeValue`, other.

percentRandomSelection

Description

The percent of the time the learning module presents a random offer. For example, setting percentRandomSelection to 5 means that 5% of the time (5 out of every 100 recommendations), the learning module presents a random offer, independent of the score. Enabling `percentRandomSelection` overrides the `offerTieBreakMethod` configuration property. When `percentRandomSelection` is enabled, this property is set regardless if learning is on or off or if built-in or external learning is used.

Default value

5

Valid Values

Any integer from 0 (which disables the `percentRandomSelection` feature) up to 100.

recencyWeightingFactor

Description

The decimal representation of a percentage of the set of data defined by the `recencyWeightingPeriod`. For example, the default value of .15 means that 15% of the data used by the learning utility comes from the `recencyWeightingPeriod`.

This property is applicable if the `Interact > offerserving > optimizationType` property for Interact runtime is set to `BuiltInLearning` only.

Default value

0.15

Valid Values

A decimal value less than 1.

recencyWeightingPeriod

Description

The size in hours of data granted the `recencyWeightingFactor` percentage of weight by the learning module. For example, the default value of 120 means that the `recencyWeightingFactor` of the data used by the learning module comes from the last 120 hours.

This property is applicable only if `optimizationType` is set to `builtInLearning`.

Default value

120

minPresentCountThreshold

Description

The minimum number of times an offer must be presented before its data is used in calculations and the learning module enters the exploration mode.

Default value

0

Valid Values

An integer greater than or equal to zero.

enablePruning

Description

If set to `Yes`, the Interact learning module algorithmically determines when a learning attribute (standard or dynamic) is not predictive. If a learning attribute is not predictive, the learning module will not consider that attribute when determining the weight for an offer. This continues until the learning module aggregates learning data.

If set to `No`, the learning module always uses all learning attributes. By not pruning non-predictive attributes, the learning module may not be as accurate as it could be.

Default value

Yes

Valid Values

Yes | No

Campaign | partitions | partition[n] | Interact | learning | learningAttributes | [learningAttribute]

These configuration properties define the learning attributes.

attributeName

Description

Each attributeName is the name of a visitor attribute you want the learning module to monitor. This must match the name of a name-value pair in your session data.

This property is applicable if the Interact > offerserving > optimizationType property for Interact runtime is set to BuiltInLearning only.

Default value

No default value defined.

Campaign | partitions | partition[n] | Interact | deployment

These configuration properties define deployment settings.

chunkSize

Description

The maximum size of fragmentation in KB for each Interact deployment package.

Default value

500

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | serverGroups | [serverGroup]

These configuration properties define server group settings.

serverGroupName

Description

The name of the Interact runtime server group. This is the name that appears on the interactive channel summary tab.

Default value

No default value defined.

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | serverGroups | [serverGroup] | instanceURLs | [instanceURL]

These configuration properties define the Interact runtime servers.

instanceURL

Description

The URL of the Interact runtime server. A server group can contain several Interact runtime servers; however, each server must be created under a new category.

Default value

No default value defined.

Example

`http://server:port/interact`

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | flowchart

These configuration properties define the Interact runtime environment used for test runs of interactive flowcharts.

serverGroup

Description

The name of the Interact server group Campaign uses to execute a test run. This name must match the category name you create under `serverGroups`.

Default value

No default value defined.

Availability

This property is applicable only if you have installed Interact.

dataSource

Description

Use the `dataSource` property to identify the physical data source for Campaign to use when performing test runs of interactive flowcharts. This property should match the data source defined by the `Campaign > partitions > partitionN > dataSources` property for the test run data source defined for Interact design time.

Default value

No default value defined.

Availability

This property is applicable only if you have installed Interact.

eventPatternPrefix

Description

The `eventPatternPrefix` property is a string value that is prepended to event pattern names to allow them to be used in expressions in Select or Decision processes within interactive flowcharts.

Note that if you change this value, you must deploy global changes in the interactive channel for this updated configuration to take effect.

Default value

EventPattern

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | whiteList | [AudienceLevel] | DefaultOffers

These configuration properties define the default cell code for the default offers table. You need to configure these properties only if you are defining global offer assignments.

DefaultCellCode

Description

The default cell code Interact uses if you do not define a cell code in the default offers table.

Default value

No default value defined.

Valid Values

A string that matches the cell code format defined in Campaign

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | whiteList | [AudienceLevel] | offersBySQL

These configuration properties define the default cell code for the offersBySQL table. You need to configure these properties only if you are use SQL queries to get a desired set of candidate offers.

DefaultCellCode

Description

The default cell code Interact uses for any offer in the OffersBySQL table(s) that has a null value in the cell code column (or if the cell code column is missing altogether. This value must be a valid cell code.

Default value

No default value defined.

Valid Values

A string that matches the cell code format defined in Campaign

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | whiteList | [AudienceLevel] | ScoreOverride

These configuration properties define the default cell code for the score override table. You need to configure these properties only if you are defining individual offer assignments.

DefaultCellCode

Description

The default cell code Interact uses if you do not define a cell code in the score override table.

Default value

No default value defined.

Valid Values

A string that matches the cell code format defined in Campaign

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | server | internal

Properties in this category specify integration settings and the internalID limits for the selected Campaign partition. If your Campaign installation has multiple partitions, set these properties for each partition that you want to affect.

internalIdLowerLimit

Configuration category

`Campaign|partitions|partition[n]|server|internal`

Description

The `internalIdUpperLimit` and `internalIdLowerLimit` properties constrain the Campaign internal IDs to be within the specified range. Note that the values are inclusive: that is, Campaign may use both the lower and upper limit.

Default value

`0` (zero)

internalIdUpperLimit

Configuration category

`Campaign|partitions|partition[n]|server|internal`

Description

The `internalIdUpperLimit` and `internalIdLowerLimit` properties constrain the Campaign internal IDs to be within the specified range. The values are inclusive: that is, Campaign may use both the lower and upper limit. If Distributed Marketing is installed, set the value to `2147483647`.

Default value

`4294967295`

eMessageInstalled

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

Indicates that eMessage is installed. When you select Yes, eMessage features are available in the Campaign interface.

The IBM installer sets this property to Yes for the default partition in your eMessage installation. For additional partitions where you installed eMessage, you must configure this property manually.

Default value

No

Valid Values

Yes | No

interactInstalled

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

After installing the Interact design environment, this configuration property should be set to Yes to enable the Interact design environment in Campaign.

If Interact is not installed, set to No. Setting this property to No does not remove Interact menus and options from the user interface. To remove menus and options, you must manually unregister Interact using the configTool utility.

Default value

No

Valid Values

Yes | No

Availability

This property is applicable only if you installed Interact.

MO_UC_integration

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

Enables integration with Marketing Operations for this partition, if the integration is enabled in the **Platform** configuration settings. For more information, see the *IBM Marketing Operations and Campaign Integration Guide*.

Default value

No

Valid Values

Yes | No

MO_UC_BottomUpTargetCells

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

For this partition, allows bottom-up cells for Target Cell Spreadsheets, if **MO_UC_integration** is enabled. When set to Yes, both top-down and bottom-up target cells are visible, but bottom-up target cells are read-only. For more information, see the *IBM Marketing Operations and Campaign Integration Guide*.

Default value

No

Valid Values

Yes | No

Legacy_campaigns

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

For this partition, enables access to campaigns created before Marketing Operations and Campaign were integrated. Applies only if **MO_UC_integration** is set to Yes. Legacy campaigns also include campaigns created in Campaign 7.x and linked to Plan 7.x projects. For more information, see the *IBM Marketing Operations and Campaign Integration Guide*.

Default value

No

Valid Values

Yes | No

IBM Marketing Operations - Offer integration

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

Enables the ability to use Marketing Operations to perform offer lifecycle management tasks on this partition, if **MO_UC_integration** is enabled for this partition. Offer integration must be enabled in your **Platform** configuration settings. For more information, see the *IBM Marketing Operations and Campaign Integration Guide*.

Default value

No

Valid Values

Yes | No

UC_CM_integration

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

Enables Digital Analytics online segment integration for a Campaign partition. If you set this value to Yes, the Select process box in a flowchart provides the option to select **Digital Analytics Segments** as input. To configure the Digital Analytics integration for each partition, choose **Settings > Configuration > Campaign | partitions | partition[n] | Coremetrics**.

Default value

No

Valid Values

Yes | No

numRowsReadToParseDelimitedFile

Configuration category

Campaign|partitions|partition[n]|server|internal

Description

This property is used when mapping a delimited file as a user table. It is also used by the Score process box when importing a score output file from IBM SPSS® Modeler Advantage Enterprise Marketing Management Edition. To import or map a delimited file, Campaign needs to parse the file to identify the columns, data types (field types), and column widths (field lengths).

The default value of 100 means Campaign examines the first 50 and the last 50 line entries in the delimited file. Campaign then allocates the field length based on the largest value it finds within those entries. In most cases, the default value is sufficient to determine field lengths. However, in very large delimited files, a later field might exceed the estimated length that Campaign computes, which can cause an error during flowchart runtime. Therefore, if you are mapping a very large file, you can increase this value to make Campaign examine more line entries. For example, a value of 200 makes Campaign examine the first 100 line entries and the last 100 line entries of the file.

A value of 0 examines the entire file. Typically, this is necessary only if you are importing or mapping files that have variable data widths of fields which cannot be identified by reading the first and last few lines. Reading the entire file for extremely large files can increase the required processing time for table mapping and Score process box runs.

Default value

100

Valid Values

0 (all lines) or any positive integer

Campaign | monitoring

Properties in the this category specify whether the Operational Monitoring feature is enabled, the URL of the Operational Monitoring server, and caching behavior. Operational Monitoring displays and allows you to control active flowcharts.

cacheCleanupInterval

Description

The `cacheCleanupInterval` property specifies the interval, in seconds, between automatic cleanups of the flowchart status cache.

This property is not available in versions of Campaign earlier than 7.0.

Default value

`600` (10 minutes)

cacheRunCompleteTime

Description

The `cacheRunCompleteTime` property specifies the amount of time, in minutes, that completed runs are cached and display on the Monitoring page.

This property is not available in versions of Campaign earlier than 7.0.

Default value

`4320`

monitorEnabled

Description

The `monitorEnabled` property specifies whether the monitor is turned on.

This property is not available in versions of Campaign earlier than 7.0.

Default value

`FALSE`

Valid values

`TRUE | FALSE`

serverURL

Description

The `Campaign > monitoring > serverURL` property specifies the URL of the Operational Monitoring server. This is a mandatory setting; modify the value if the Operational Monitoring server URL is not the default.

If Campaign is configured to use Secure Sockets Layer (SSL) communications, set the value of this property to use HTTPS. For example: `serverURL=https://`*`host:SSL_port`*`/Campaign/OperationMonitor` where:

- *`host`* is the name or IP address of the machine on which the web application is installed
- *`SSL_Port`* is the SSL port of the web application.

Note the `https` in the URL.

Default value

`http://localhost:7001/Campaign/OperationMonitor`

monitorEnabledForInteract

Description

If set to `TRUE`, enables Campaign JMX connector server for Interact. Campaign has no JMX security.

If set to `FALSE`, you cannot connect to the Campaign JMX connector server.

This JMX monitoring is for the Interact contact and response history module only.

Default value

`FALSE`

Valid Values

`TRUE | FALSE`

Availability

This property is applicable only if you have installed Interact.

protocol

Description

Listening protocol for the Campaign JMX connector server, if `monitorEnabledForInteract` is set to yes.

This JMX monitoring is for the Interact contact and response history module only.

Default value

JMXMP

Valid Values

JMXMP | RMI

Availability

This property is applicable only if you have installed Interact.

port

Description

Listening port for the Campaign JMX connector server, if `monitorEnabledForInteract` is set to yes.

This JMX monitoring is for the Interact contact and response history module only.

Default value

2004

Valid Values

An integer between 1025 and 65535.

Availability

This property is applicable only if you have installed Interact.

Campaign | partitions | partition[n] | Interact | outboundChannels

These configuration properties enable you to tune the outbound channels for triggered messages.

category name

Description

This property defines the name of this outbound channel. The name must be unique among all outbound channels.

name

Description

The name of your outbound channel.

Note: You must restart your application server for the changes to take effect.

Campaign | partitions | partition[n] | Interact | outboundChannels | Parameter Data

These configuration properties enable you to tune the outbound channels for triggered messages.

cateogry name

Description

This property defines the name of this parameter. The name must be unique among all parameters for that outbound channnel.

value

Description

This property defines the parameters, in the format of name value pairs, needed by this outbound gateway.

Campaign | partitions | partition[n] | Interact | Simulator

These configuration properties define the server group you want to use to run API simulations.

serverGroup

Description

Specify the runtime server group that is used to run API simulations.

Default value

defaultServerGroup

Chapter 16. Real-time offer personalization on the client side

There may be situations where you want to provide real-time offer personalization without implementing low-level Java code or SOAP calls to the Interact server. For example, when a visitor initially loads a web page where Javascript content is your only extended programming available, or when a visitor opens an email message where only HTML content is possible. IBM Interact provides several connectors that provide real-time offer management in situations where you have control only over the web content that is loaded on the client side, or where you want to simplify your interface to Interact.

Your Interact installation includes two connectors for offer personalization that is initiated on the client side:

- "About the Interact Message Connector." Using the Message Connector, web content in email messages (for example) or other electronic media can contain image and link tags to make calls to the Interact server for page-load offer presentation and click-through landing pages.
- "About the Interact Web Connector" on page 306. Using the Web Connector (also called the JS Connector) web pages can use client-side JavaScript to manage offer arbitration, presentation, and contact/response history through page-load offer presentation and click-through landing pages.

About the Interact Message Connector

The Interact Message Connector allows email messages and other electronic media to make calls to IBM Interact to allow personalized offers to be presented at open-time, and when the customer clicks-through the message to the specified site. This is accomplished through the use of two key tags: The image tag (`IMG`), which loads the personalized offers at open-time, and the link tag (`A`), which captures information about click-through and redirects the customer to a specific landing page.

Example

The following example shows some HTML code that you might include in a marketing spot (for example, within an email message) that contains both an `IMG` tag URL (which passes information when the document opens to the Interact server and retrieves the appropriate offer image in response) and an `A` tag URL (which determines what information is passed to the Interact server on click-through):

```
<a href="http://www.example.com/MessageConnector/
offerClickthru.jsp?msgId=1234&linkId=1&userid=1&referral=test"><img
src="http:// www.example.com/MessageConnector/offerImage.jsp?msgId=1234
&linkId=1&userid=1&incomeLevel=5&incomeType=numeric"/></a>
```

In the following example, an `IMG` tag is enclosed in an `A` tag, causes the following behavior:

1. When the email message is opened, the Message Connector receives a request containing the information encoded in the `IMG` tag: the `msgID` and `linkID` for this message, and customer parameters that include userid, income level, and income type.

2. This information is passed through an API call to the Interact runtime server.
3. The runtime server returns an offer to the Message Connector, which retrieves the URL of the offer image, and provides that URL (with any additional parameters included) and redirects the image request to that offer URL.
4. The customer sees the offer as an image.

At that point, the customer may click that image to respond to the offer in some way. That click-through, using the `A` tag and its specified `HREF` attribute (which specifies the destination URL) sends another request to the Message Connector for a landing page linked to that offer's URL. The customers browser is then redirected to the landing page as configured in the offer.

Note that a click-through `A` tag is not strictly necessary; the offer may consist of an image only, such as a coupon for the customer to print.

Installing the Message Connector

The files you require to install, deploy, and run the Message Connector have automatically been included with your IBM Interact runtime server installation. This section summarizes the steps needed to get the Message Connector ready for use.

Installing and deploying the Message Connector involves the following tasks:

- Optionally, configuring the default settings for the Message Connector as described in "Configuring the Message Connector."
- Creating the database tables needed to store the Message Connector transaction data as described in "Creating the Message Connector Tables" on page 302.
- Installing the Message Connector web application as described in "Deploying and running the Message Connector" on page 303.
- Creating the `IMG` and `A` tags in your marketing spots (email or web pages, for example) needed to call Message Connector offers on open and click-through, as described in "Creating the Message Connector links" on page 304.

Configuring the Message Connector

Before you deploy the Message Connector, you must customize the configuration file included with your installation to match your specific environment. You can modify the XML file called `MessageConnectorConfig.xml` found in your Message Connector directory on the Interact runtime server, similar to `<Interact_home>/msgconnector/config/MessageConnectorConfig.xml`.

The `MessageConnectorConfig.xml` file contains some configuration settings that are required, and some that are optional. The settings that you use must be customized for your specific installation. Follow the steps here to modify the configuration.

1. If the Message Connector is deployed and running on your web application server, undeploy the Message Connector before continuing.
2. On the Interact runtime server, open the `MessageConnectorConfig.xml` file in any text or XML editor.
3. Modify the configuration settings as needed, making sure that the following *required* settings are correct for your installation.
   - `<interactUrl>`, the URL of the Interact runtime server to which the Message Connector page tags should connect, and on which the Message Connector is running.
   -

<imageErrorLink>, the URL to which the Message Connector will redirect if an error occurs while processing a request for an offer image.

- <landingPageErrorLink>, the URL to which the Message Connector will redirect if an error occurs while processing a request for an offer landing page.
- <audienceLevels>, a section of the configuration file that contains one or more set of audience level settings, and which specifies the default audience level if none is specified by the Message Connector link. There must be at least one audience level configured.

All of the configuration settings are described in greater detail in "Message Connector Configuration Settings."

4. When you have completed the configuration changes, save and close the MessageConnectorConfig.xml file.
5. Continue setting up and deploying the Message Connector.

Message Connector Configuration Settings:

To configure the Message Connector, you can modify the XML file called MessageConnectorConfig.xml found in your Message Connector directory on the Interact runtime server, usually <Interact_home>/msgconnector/config/MessageConnectorConfig.xml. Each of the configurations in this XML file are described here. Be aware that if you modify this file after the Message Connector is deployed and running, be sure to undeploy and redeploy the Message Connector or restart the application server to reload these settings after you are done modifying the file.

General Settings

The following table contains a list of the optional and required settings contained in the generalSettings section of the MessageConnectorConfig.xml file.

Table 24. Message Connector General Settings

Element	Description	Default Value
<interactURL>	The URL of the Interact runtime server to handle the calls from the Message Connector page tags, such as the runtime server on which the Message Connector is running. This element is required.	http://localhost:7001/interact
<defaultDateTimeFormat>	The default date format.	MM/dd/yyyy
<log4jConfigFileLocation>	The location of the Log4j property file. It is relative to *$MESSAGE_CONNECTOR_HOME* environment variable if it is set; otherwise, this value is relative to the root path of Message Connector web application.	config/MessageConnectorLog4j.properties

Default Parameter Values

The following table contains a list of the optional and required settings contained in the `defaultParameterValues` section of the `MessageConnectorConfig.xml` file.

Table 25. Message Connector Default Parameter Settings

Element	Description	Default Value
`<interactiveChannel>`	The name of the default interactive channel.	
`<interactionPoint>`	The name of the default interaction point.	
`<debugFlag>`	Determines whether debugging is enabled. The allowed values are `true` and `false`.	`false`
`<contactEventName>`	The default name of the contact event that is posted.	
`<acceptEventName>`	The default name of the accept event that is posted.	
`<imageUrlAttribute>`	The default offer attribute name that contains the URL for the offer image, if none is specified in the Message Connector link.	
`<landingPageUrlAttribute>`	The default URL for the click-through landing page if none is specified in the Message Connector link.	

Behavior Settings

The following table contains a list of the optional and required settings contained in the `behaviorSettings` section of the `MessageConnectorConfig.xml` file.

Table 26. Message Connector Behavior Settings

Element	Description	Default Value
`<imageErrorLink>`	The URL to which the connector redirects if an error occurs while processing a request for an offer image. This setting is required.	`/images/default.jpg`
`<landingPageErrorLink>`	The URL to which the connector redirects if an error occurs while processing a request for a click-through landing page. This setting is required.	`/jsp/default.jsp`
`<alwaysUseExistingOffer>`	Determines whether the cached offer should be returned, even if it already expired. The allowed values are `true` and `false`.	`false`
`<offerExpireAction>`	The action to take if the original offer is found, but is already expired. Allowed values are: • `GetNewOffer` • `RedirectToErrorPage` • `ReturnExpiredOffer`	`RedirectToErrorPage`

Storage Settings

The following table contains a list of the optional and required settings contained in the storageSettings section of the MessageConnectorConfig.xml file.

Table 27. MessageConnector Storage Settings

Element	Description	Default Value
<persistenceMode>	When the cache persists new entries into the database. The allowed values are WRITE-BEHIND (where data is written to the cache initially, and updated to the database at a later time) and WRITE-THROUGH (where data is written to the cache and to the database at the same time).	WRITE-THROUGH
<maxCacheSize>	The maximum number of entries in the memory cache.	5000
<maxPersistenceBatchSize>	The maximum batch size while persisting entries into the database.	200
<macCachePersistInterval>	The maximum time in seconds an entry stays in the cache before it is persisted into the database.	3
<maxElementOnDisk>	The maximum number of entries in the disk cache.	5000
<cacheEntryTimeToExpireInSeconds>	The maximum amount of time for the entries in the disk cache to persist before expiring.	60000
<jdbcSettings>	A section of the XML file containing specific information if a JDBC connection is used. It is mutually exclusive with the <dataSourceSettings> section.	Configured by default to connect to a SQLServer database configured on the local server, but if you enable this section you must provide the actual JDBC settings and credentials to log in.
<dataSourceSettings>	A section of the XML file containing specific information if a data source connection is used. It is mutually exclusive with the <jdbcSettings> section.	Configured by default to connect to the InteractDS data source defined on the local web application server.

Audience Levels

The following table contains a list of the optional and required settings contained in the audienceLevels section of the MessageConnectorConfig.xml file.

Note that the audienceLevels element is optionally used to specify the default audience level to use if none is specified in the Message Connector link, as in the following example:

```
<audienceLevels default="Customer">
```

In this example, the value for the default attribute matches the name of an audienceLevel defined in this section. There must be at least one audience level defined in this configuration file.

Table 28. MessageConnector Audience Level Settings

Element	Element	Description	Default Value
<audienceLevel>		The element containing the audience level configuration. Provide a name attribute, as in <audienceLevel name="Customer">	
	<messageLogTable>	The name of the log table. This value is required.	UACI_MESSAGE_CONNECTOR_LOG
<fields>	<field>	The definition of one or more audience ID fields for this audienceLevel.	
	<name>	The name of the audience ID field, as specified in the Interact runtime.	
	<httpParameterName>	The corresponding parameter name for this audience ID field.	
	<dbColumnName>	The corresponding column name in the database for this audience ID field.	
	<type>	The type of the audience ID field, as specified in the Interact runtime. Values can be string or numeric.	

Creating the Message Connector Tables

Before you can deploy the IBM Interact Message Connector, you must first create the tables in the database where the Interact runtime data is stored. You'll create one table for each audience level you have defined. For each audience level, Interact will use the tables you create to record information about Message Connector transactions.

Use your database client to run the Message Connector SQL script against the appropriate database or schema to create the necessary tables. The SQL scripts for your supported database are installed automatically when you install the Interact runtime server. See the worksheets you completed in the *IBM Interact Installation Guide* for details about connecting to the database that contains the Interact runtime tables.

1. Launch your database client and connect to the database in which your Interact runtime tables are currently stored.
2. Run the appropriate script in the <*Interact_home*>/msgconnector/scripts/ddl directory. The following table lists the sample SQL scripts you can use to manually create the Message Connector tables:

Table 29. Scripts for creating the Message Connector tables

Data source type	Script name
IBM DB2	db_scheme_db2.sql
Microsoft SQL Server	db_scheme_sqlserver.sql
Oracle	db_scheme_oracle.sql

Note that these scripts are provided as samples. You may use a different naming convention or structure for audience ID values, so you may need to modify the script before running it. In general, it is a best practice to have one table dedicated to each audience level.

The tables are created to contain the following information:

Table 30. Information created by the sample SQL scripts

Column Name	Description
LogId	The primary key of this entry.
MessageId	The unique identifier of each messaging instance.
LinkId	The unique identifier of each link in the electronic media (such as an email message).
OfferImageUrl	The URL to the related image of the returned offer.
OfferLandingPageUrl	The URL to the related landing page of the returned offer.
TreatmentCode	The treatment code of the returned offer.
OfferExpirationDate	The expiration date and time of the returned offer.
OfferContactDate	The date and time that the offer was returned to the client.
AudienceId	The audience ID of the electronic media.

Note the following about this table:

- Depending on the audience level, there will be one AudienceId column for each component of the audience key.
- The combination of MessageId, LinkId, and AudienceId(s) forms a unique key of this table.

When the script has finished running, you have created the necessary tables for the Message Connector.

You are now ready to deploy the Message Connector web application.

Deploying and running the Message Connector

The IBM Interact Message Connector is deployed as a stand-alone web application on a supported web application server.

Before you can deploy the Message Connector, be sure that the following tasks have been complete:

- You must have installed the IBM Interact runtime server. The deployable Message Connector application is automatically installed along with the runtime server, and is ready to deploy from your Interact home directory.
- You must also have run the SQL scripts provided with your installation to create the necessary tables in the Interact runtime database for use by the Message Connector as described in "Creating the Message Connector Tables" on page 302

Just as you deploy other IBM applications on a web application server before you can run them, you must deploy the Message Connector application to make it available for serving offers.

1. Connect to your web application server management interface with the necessary privileges to deploy an application.
2. Follow the instructions for your web application server to deploy and run the file called *<Interact_home>*/msgconnector/MessageConnector.war Replace *<Interact_home>* with the actual directory into which the Interact runtime server is installed.

The Message Connector is now available for use. After you have configured your Interact installation to create the underlying data that the Message Connector will use to provide offers, such as interactive channels and strategies, flowcharts, offers, and so on, you can create the links in your electronic media that the Message Connector will accept.

Creating the Message Connector links

To use the Message Connector to provide custom offer images when an end-user interacts with your electronic media (such as by opening an email message), and custom landing pages when the end-user clicks through the offer, you need to create the links to embed in your message. This section provides a summary of the HTML tagging of those links.

Regardless of the system you use to generate your outgoing messages to end users, you need to generate the HTML tagging to contain the appropriate fields (provided in the HTML tags as attributes) containing information you want to pass to the Interact runtime server. Follow the steps below to configure the minimum information needed for a Message Connector message.

Note that although the instructions here refer specifically to messages containing Message Connector links, you can follow the same steps and configuration to add links to web pages or any other electronic media.

1. Create the `IMG` link that will appear in your message with, at a minimum, the following parameters:
   - `msgID`, indicating the unique identifier for this message.
   - `linkID`, indicating the unique identifier for the link in the message.
   - `audienceID`, the identifier of the audience to which the recipient of the message belongs.

     Note that if the audience ID is a composite ID, all of those components must be included in the link.

   You may also include optional parameters that include audience level, interactive channel name, interaction point name, image location URL, and your own custom parameters not specifically used by the Message Connector.
2. Optionally, create an `A` link that encloses the `IMG` link so that, when the user clicks the image, the browser loads a page containing the offer for the user. The `A` link must also contain the three parameters listed above (`msgID`, `linkID`, and `audienceID`), plus any optional parameters (audience level, interactive channel name, and interactive point name) and custom parameters not specifically used by the Message Connector. Note that the `A` link will most likely contain a Message Connector `IMG` link, but can also stand alone on the page as needed. If the link does contain an `IMG` link, the `IMG` link should contain the same set of parameters as the enclosing `A` link (including any optional or custom parameters).
3. When the links are correctly defined, generate and send the email messages.

For detailed information on the available parameters, and sample links, see “"IMG" and "A" tag HTTP Request parameters”

"IMG" and "A" tag HTTP Request parameters

When Message Connector receives a request, either because an end user opened an email containing a Message Connector-encoded `IMG` tag or because the end user clicked-through an `A` tag, it parses the parameters included with the request to return the appropriate offer data. This section provides a list of the parameters that can be included in the requesting URL (either the `IMG` tag (loaded automatically when a tagged image is displayed when the email is opened) or the `A` tag (loaded when the person viewing the email clicks through the message to the specified site).

Parameters

When Message Connector receives a request, it parses the parameters included with the request. These parameters include some or all of the following:

Parameter Name	Description	Required?	Default Value
`msgId`	The unique identifier of the email instance or web page.	Yes	None. This is provided by the system creating the unique instance of the email message or web page containing the tag.
`linkId`	The unique identifier of the link in this email or web page.	Yes	None. This is provided by the system creating the unique instance of the email message or web page containing the tag.
`audienceLevel`	The audience level to which the recipient of this communication belongs.	No	The audienceLevel specified as the default in the `audienceLevels` element found in the MessageConnectorConfig.xml file.
`ic`	The name of the target interactive channel (IC)	No	The value of the `interactiveChannel` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file, which is "interactiveChannel" by default.
`ip`	The name of the applying interaction point (IP)	No	The value of the `interactionPoint` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file, which is "headBanner" by default.
`offerImageUrl`	The URL of the target offer image for the IMG URL in the message.	No	None.
`offerImageUrlAttr`	The name of the offer attribute that has the URL of the target offer image	No	The value of the `imageUrlAttribute` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file.
`offerLandingPageUrl`	The URL of the landing page corresponding to the target offer.	No	None.
`offerLandingPageUrlAttr`	The name of the offer attribute that has the URL of the landing page corresponding to the target offer.	No	The value of the `landingPageUrlAttribute` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file.
`contactEvent`	The name of the contact event.	No	The value of the `contactEventName` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file, which is "contact" by default.
`responseEvent`	The name of the accept event.	No	The value of the `acceptEventName` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file, which is "accept" by default.
`debug`	The debug flag. Set this parameter to "true" only for troubleshooting and at the instruction of IBM technical support.	No	The value of the `debugFlag` element found in the `defaultParameterValues` section of the MessageConnectorConfig.xml file, which is "false" by default.
<audience id>	The audience ID of this user. The name of this parameter is defined in the configuration file.	Yes	None.

When the Message Connector receives a parameter that is unrecognized (that is, does not appear in the above list), it is handled in one of two possible ways:

- If an unrecognized parameter is provided (for example, "attribute", as in `attribute="attrValue"`) and there is a matching parameter of the same name plus the word "Type" (for example, "attributeType", as in `attributeType="string"`), this causes the Message Connector to create a matching Interact parameter and pass it to the Interact runtime.

The values for the Type parameter can be any of the following:

– string
– numeric
– datetime

  For a parameter of type "datetime," the Message Connector also looks for a parameter of the same name plus the word "Pattern" (for example, "attributePattern") whose value is a valid date/time format. For example, you might provide the parameter `attributePattern="MM/dd/yyyy"`.

  Note that if you specify a parameter type of "datetime" but do not provide a matching date pattern, the value specified in the Message Connector configuration file (found in `<installation_directory>/msgconnector/config/MessageConnectorConfig.xml`) on the Interact server is used.

- If an unrecognized parameter is provided and there is no matching Type value, Message Connector passes that parameter to the target redirect URL.

For all unrecognized parameters, the Message Connector passes them to the Interact runtime server without processing or saving them.

Example Message Connector Code

The following `A` tag contains an example of a set of Message Connector links that might appear in an email message:

```
<a href="http://www.example.com/MessageConnector/offerClickthru.jsp?msgId=234
   &linkId=1&userid=1&referral=xyz">
 <img src="http://www.example.com/MessageConnector/offerImage.jsp?msgId=234&linkId=1
      &userid=1&incomeCode=3&incomeType=numeric"/>
</a>
```

In this example, the `IMG` tag loads automatically when the email message is opened. By retrieving the image from the specified page, the message passes the parameters for the unique message identifier (msgID), unique link identifier (linkID), and unique user identifier (userid), along with two additional parameters (incomeCode and incomeType) to be passed to the Interact runtime.

The `A` tag provides the HREF (Hypertext Reference) attribute that turns the offer image into a clickable link in the email message. If the viewer of the message, upon seeing the image, clicks through to the landing page, the unique message identifier (`msgId`), link identifier (`linkId`), and user identifier (`userid`) are passed through to the server, as well as one additional parameter (`referral`) that is passed to the target redirect URL.

About the Interact Web Connector

The Interact WebConnector (also referred to as the JavaScript Connector, or JSConnector) provides a service on the Interact runtime server that allows JavaScript code to call the Interact Java API. This enables web pages to make calls to Interact for real-time offer personalization using only embedded JavaScript code, without having to rely on web development languages (such as Java, PHP, JSP, and so on). For example, you might embed a small snippet of JavaScript code on each page of your web site that serve offers recommended by Interact, so that each time the page loads, calls are made to the Interact API to ensure that the best offers are displayed on the loading page for the site visitor.

Use the Interact Web Connector in situations where you want to display offers to visitors on a page where you may not have server-side programmatic control over

the page display (as you would with, for example, PHP or other server-based scripting), but can still embed JavaScript code in your page content that will be executed by the visitor's web browser.

Tip: The Interact Web Connector files are installed automatically onto your Interact runtime server, in the directory <*Interact_home*>/jsconnector. In the directory <*Interact_home*>/jsconnector, you'll find a ReadMe.txt containing important notes and details about the Web Connector features, as well as sample files and the Web Connector source code to use as the basis for developing your own solutions. If you do not find information to answer your questions here, see the jsconnector directory for more information.

Installing the Web Connector on the runtime server

An instance of the Web Connector is installed automatically with your IBM Interact runtime server, and is enabled by default. However, there are some settings you must modify before you can configure and use the Web Connector.

The settings you must modify before you can use the Web Connector that is installed on the runtime server are added to the web application server's configuration. For that reason, you will need to restart the web application server after completing these steps.

1. For the web application server on which the Interact runtime server is installed, set the following Java properties:

   -DUI_JSCONNECTOR_ENABLE_INPROCESS=true

   -DUI_JSCONNECTOR_HOME=<*jsconnectorHome*>

   Replace <*jsconnectorHome*> with the path to the jsconnector directory on the runtime server, which is <Interact_Home>/jsconnector.

   The way in which you set the Java properties depends on your web application server. For example, in WebLogic, you would edit the startWebLogic.sh or startWebLogic.cmd file to update the JAVA_OPTIONS setting, as in this example:

   ```
   JAVA_OPTIONS="${SAVE_JAVA_OPTIONS} -DUI_JSCONNECTOR_HOME=/UnicaFiles/
   jsconnector"
   ```

   In WebSphere Application Server, you would set this property in the Java virtual machine panel of the administration console.

   See your web application server documentation for specific instructions on setting Java properties.
2. Restart your web application server if it was already running, or start your web application server now, to make sure that the new Java properties are used.

When the web application server has completed its startup process, you have finished installing the Web Connector on the runtime server. The next step is to connect to its Configuration web page at http://<*host*>:<*port*>/interact/jsp/WebConnector.jsp, where <*host*> is the Interact runtime server name, and <*port*> is the port on which the Web Connector is listening, as specified by the web application server.

Installing the Web Connector as a separate web application

An instance of the Web Connector is installed automatically with your IBM Interact runtime server, and is enabled by default. However, you can also deploy the Web Connector as its own web application (for example, in a web application server on a separate system) and configure it to communicate with the remote Interact runtime server.

These instructions describe the process of deploying the Web Connector as a separate web application with access to a remote Interact runtime server.

Before you can deploy the Web Connector, you must have installed the IBM Interact runtime server, and you must have a web application server on another system with network access (not blocked by any firewall) to the Interact runtime server.

1. Copy the `jsconnector` directory containing the Web Connector files from your Interact runtime server to the system where the web application server (such as WebSphere Application Server) is already configured and running. You can find the `jsconnector` directory in your Interact installation director.
2. On the system where you'll be deploying the Web Connector instance, configure the `jsconnector/jsconnector.xml` file using any text or XML editor to modify the `interactURL` attribute.

   This is set by default to `http://localhost:7001/interact`, but you must modify it to match the URL of the remote Interact runtime server, such as `http://runtime.example.com:7011/interact`.

   After you deploy the Web Connector, you can use a web interface to customize the remaining settings in the `jsconnector.xml` file. See "Configuring the Web Connector" on page 309 for more information.
3. For the web application server on which you will be deploying the Web Connector, set the following Java property:

   `-DUI_JSCONNECTOR_HOME=<jsconnectorHome>`

   Replace *<jsconnectorHome>* with the actual path to where you copied the `jsconnector` directory onto the web application server.

   The way in which you set the Java properties depends on your web application server. For example, in WebLogic, you would edit the `startWebLogic.sh` or `startWebLogic.cmd` file to update the `JAVA_OPTIONS` setting, as in this example:

   ```
   JAVA_OPTIONS="${SAVE_JAVA_OPTIONS} -DUI_JSCONNECTOR_HOME=/InteractFiles/
   jsconnector"
   ```

   In WebSphere Application Server, you would set this property in the Java virtual machine panel of the administration console.

   See your web application server documentation for specific instructions on setting Java properties.
4. Restart your web application server if it was already running, or start your web application server in this step, to make sure that the new Java property is used.

   Wait for the web application server to complete its startup process before continuing.
5. Connect to your web application server management interface with the necessary privileges to deploy an application.
6. Follow the instructions for your web application server to deploy and run the following file: `jsConnector/jsConnector.war`

The Web Connector is now deployed in the web application. After you have your fully-configured Interact server up and running, the next step is to connect to the Web Connector Configuration web page at `http://` *<host>*`:` *<port>*`/interact/jsp/WebConnector.jsp`, where *<host>* is the system running the web application server on which you just deployed the Web Connector, and *<port>* is the port on which the Web Connector is listening, as specified by the web application server.

Configuring the Web Connector

Configuration settings for the Interact Web Connector are stored in a file called `jsconnector.xml` that is stored on the system where the Web Connector is deployed (such as the Interact runtime server itself, or a separate system running a web application server). You can edit the `jsconnector.xml` file directly using any text editor or XML editor; however, an easier way to configure almost all of the available configuration settings is to use the Web Connector Configuration page from your web browser.

Before you can use the web interface to configure the Web Connector, you must install and deploy the web application that provides the Web Connector. On the Interact runtime server, an instance of the Web Connector is installed automatically when you install and deploy Interact. On any other web application server, you must install and deploy the Web Connector web application as described in "Installing the Web Connector as a separate web application" on page 307.

1. Open your supported web browser and open a URL similar to the following:

   `http://<host>:<port>/interact/jsp/WebConnector.jsp`

   - Replace *<host>* with the server on which the Web Connector is running, such as the host name of the runtime server or the name of the server on which you deployed a separate instance of the Web Connector.
   - Replace *<port>* with the port number on which the Web Connector web application is listening for connections, which usually matches the web application server's default port.
2. On the Configurations page that appears, complete the following sections:

Table 31. Web Connector Configuration Settings Summary.

Section	Settings
Basic Settings	Use the Basic Settings page to configure the overall Web Connector behavior for the site on which you'll be rolling out the tagged pages. These settings include the base URL for the site, information that Interact needs to use about the visitors to the site, and similar settings that apply to all of the pages you plan to tag with Web Connector code. See "WebConnector Configuration Basic Options" on page 311 for details.
HTML Display Types	Use the HTML Display Types page to determine the HTML code that will be provided for each interaction point on the page. You can choose from the list of default templates (.flt files) that contain some combination of cascading style sheet (CSS) code, HTML code, and Javascript code to use for each interaction point. You can use the templates as provided, customize them as needed, or create your own. Configuration settings on this page correspond to the `interactionPoints` section of the `jsconnector.xml` configuration file. See "WebConnector Configuration HTML Display Types" on page 312 for details.

Table 31. Web Connector Configuration Settings Summary (continued).

Section	Settings
Enhanced Pages	Use the Enhanced Pages page to map page-specific settings to a URL pattern. For example, you might set up a page mapping such that any URL containing the text "index.htm" displays your general welcome page, with specific page load events and interaction points defined for that mapping. Configuration settings on this page correspond to the `pageMapping` section of the `jsconnector.xml` configuration file. See "WebConnector Configuration Enhanced Pages" on page 315 for details.

3. On the Basic Settings page, verify that the site-wide settings are valid for your installation, and optionally specify debug mode (not recommended unless you are troubleshooting a problem), Digital Analytics for On Premises Page Tag integration, and the default settings for most Interaction Points, then click the HTML Display Types link under Configurations.
4. On the HTML Display types page, follow these steps to add or modify display templates that define the interaction points on the customer web page.

   By default, display templates (.flt files) are stored in `<jsconnector_home>/conf/html`.

   a. Select the .flt file from the list that you want to examine or use as your starting point, or click Add a Type to create a new, blank Interaction Point template to use.

      Information about the template's contents, if any, appears next to the template list.

   b. Optionally, modify the name of the template in the **File name for this display type** field. For a new template, update `CHANGE_ME.flt` to something more meaningful.

      If you rename the template here, the Web Connector creates a new file with that name the next time the template is saved. Templates are saved when you modify the body of the text, and then navigate to any other field.

   c. Modify or complete the HTML Snippet information as needed, including any stylesheet (CSS), JavaScript, and HTML code you want to include. Note that you can also include variables that will be replaced by Interact parameters at runtime. For example, `${offer.HighlightTitle}` is automatically replaced by the offer title in the specified location of the Interaction Point.

      Use the examples that appear below the HTML Snippet field for indications of how to format your CSS, JavaScript, or HTML code blocks.
5. Use the Enhanced Pages page as needed to set up the page mappings that determine how specific URL patterns are handled on the pages.
6. When you have finished setting the configuration properties, click **Roll Out the Changes**. Clicking **Roll Out the Changes** performs the following actions:
   - Displays the IBM Interact Web Connector page tag, which contains the JavaScript code that you can copy from the Web Connector page and insert onto your web pages.
   - Backs up the existing Web Connector configuration file on the Interact server (the `jsconnector.xml` file on the server where the Web Connector is installed) and creates a new configuration file with the settings you've defined.

     Backup configuration files are stored in `<jsconnector_home>/conf/archive/jsconnector.xml.<date>.<time>`, as in

jsconnector.xml.20111113.214933.750-0500 (where the date string is 20111113 and the time string, including the time zone indicator, is 214933.750-0500)

You have now completed configuring the Web Connector.

To modify your configuration, you can either return to the beginning of these steps and perform them again with new values, or you can open the configuration file (<Interact_home>/jsconnector/conf/jsconnector.xml) in any text or XML editor and modify it as needed.

WebConnector Configuration Basic Options

Use the Basic Settings page of the Web Connector Configurations page to configure the overall Web Connector behavior for the site on which you'll be rolling out the tagged pages. These settings include the base URL for the site, information that Interact needs to use about the visitors to the site, and similar settings that apply to all of the pages you plan to tag with Web Connector code.

Site-wide Settings

The Site-wide Settings configuration options are global settings that affect the overall behavior of the installation of the Web Connector you are configuring. You can specify the following values:

Table 32. Site-wide settings for the Web Connector installation

Setting	Description	Equivalent setting in jsconnector.xml
Interact API URL	The base URL of the Interact runtime server. **Note:** This setting is used only if the Web Connector is not running inside of the Interact runtime server (that is, you have deployed it separately).	*<interactURL>*
Web Connector URL	The base URL used to generate the click-through URL.	*<jsConnectorURL>*
Interactive Channel name for the target website	The name of the interactive channel you have defined on the Interact server that represents this page mapping.	*<interactiveChannel>*
Audience Level of Visitors	The Campaign audience level for the inbound visitor; used in the API call to the Interact runtime.	*<audienceLevel>*
Audience ID Field Name in the Profile Table	Name of the audienceId field that will be used in the API call to Interact. Note that there is currently no support for multi-field audience identifiers.	*<audienceIdField>*
Data type of the Audience ID Field	The data type of the Audience ID field (either "numeric" or "string") to be used in the API call to Interact.	*<audienceIdFieldType>*
Cookie Name that represents the Session ID	The name of the cookie that will contain the session ID.	*<sessionIdCookie>*
Cookie Name that represents the Visitor ID	The name of the cookie that will contain the visitor ID.	*<visitorIdCookie>*

Optional Features

The Optional Features configuration options are optional global settings for the installation of the Web Connector you are configuring. You can specify the following values:

Table 33. Optional site-wide settings for the Web Connector installation

Setting	Description	Equivalent setting in jsconnector.xml
Enable Debug Mode	Specifies (with a `yes` or `no` answer) whether to use a special debug mode. If you enable this feature, the content returned from the Web Connector includes a Javascript call to 'alert', informing the client of the particular page mapping that just occurred. The client must have an entry in the file specified by the *<authorizedDebugClients>* setting in order to get the alert.	*<enableDebugMode>*
Authorized Debugging Clients Host File	The path to a file that contains the list of hosts or IP (Internet Protocol) addresses that qualify for debug mode. A client's host name or IP address must appear in the specified file for debug information to be collected.	*<authorizedDebugClients>*
Enable Digital Analytics for On Premises Page Tag Integration	Specifies (with a `yes` or `no` answer) whether the Web Connector should attach the specified IBM Digital Analytics for On Premises tag at the end of the page content.	`<enableNetInsightTagging>`
Digital Analytics for On Premises Tag HTML Template File	The HTML/Javascript template used to integrate a call to the Digital Analytics for On Premises tag. In general, you should accept the default setting unless you are instructed to provide a different template.	`<netInsightTag>`

WebConnector Configuration HTML Display Types

Use the HTML Display Types page to determine the HTML code that will be provided for each interaction point on the page. You can choose from the list of default templates (.flt files) that contain some combination of cascading style sheet (CSS) code, HTML code, and JavaScript code to use for each interaction point. You can use the templates as provided, customize them as needed, or create your own.

Note: Configuration settings on this page correspond to the `interactionPoints` section of the `jsconnector.xml` configuration file.

The interaction point can also contain placeholders (zones) into which offer attributes can be dropped automatically. For example, you might include `${offer.TREATMENT_CODE}`which would be replaced with the treatment code assigned to that offer during the interaction.

The templates that appear on this page are loaded automatically from the files stored in `<Interact_home>/jsconnector/conf/html` directory of the Web Connector server. Any new templates you create here are also stored in that directory.

To use the HTML Display Types page to view or modify any of the existing templates, select the `.flt` file from the list.

To create a new template on the HTML Display Types page, click **Add a Type**.

Regardless of the method you choose to create or modify a template, the following information appears next to the template list:

Setting	Description	Equivalent setting in jsconnector.xml
File name for this display type	The name assigned to the template you are editing. This name must be valid for the operating system on which the Web Connector is running; for example, you cannot use a slash (/) in the name if the operating system is Microsoft Windows. If you are creating a new template, this field is preset to `CHANGE_ME.flt`. You must change this to a meaningful value before continuing.	*<htmlSnippet>*

Setting	Description	Equivalent setting in jsconnector.xml
HTML Snippet	The specific content that Web Connector should insert into the Interaction Point on the web page. This snippet can contain HTML code, CSS formatting information, or JavaScript to be executed on the page. Each of those three types of content must be enclosed by BEGIN and END codes, as in the following examples: • ${BEGIN_HTML} <*your HTML content*> ${END_HTML} • ${BEGIN_CSS} <*your Interaction Point-specific stylesheet information*> ${END_CSS} • ${BEGIN_JAVASCRIPT} <*your Interaction Point-specific JavaScript code*> ${END_JAVASCRIPT} You can also enter a number of pre-defined special codes that are replaced automatically when the page is loaded, including the following: • `${logAsAccept}` : A macro that takes two parameters (a target URL, and the TreatmentCode used to identify the acceptance of the offer) and replaces it with the click-through URL. • `${offer.AbsoluteLandingPageURL}` • `${offer.OFFER_CODE}` • `${offer.TREATMENT_CODE}` • `${offer.TextVersion}` • `$offer.AbsoluteBannerURL}` Each of the offer codes listed here represent offer attributes defined in the offer template in IBM Campaign that was used by the marketer to create the offers that Interact is returning. Note that the Web Connector uses a template engine called FreeMarker that provides many additional options that you may find useful in setting up codes on your page templates. See http://freemarker.org/docs/index.html for more information.	No equivalent because the HTML snippet resides in its own file separate from jsconnector.xml.

Setting	Description	Equivalent setting in jsconnector.xml
Example Special Codes	Contains samples of the type of special codes, including codes that identify blocks as HTML, CSS, or JAVASCRIPT, and droppable zones you can insert to refer to specific offer metadata.	No equivalent.

Your changes to this page are saved automatically when you navigate to another Web Connector configuration page.

WebConnector Configuration Enhanced Pages

Use the Enhanced Pages page to map page-specific settings to a URL pattern. For example, you might set up a page mapping such that any incoming URL containing the text "index.htm" displays your general welcome page, with specific page load events and interaction points defined for that mapping.

Note: Configuration settings on this page correspond to the `pageMapping` section of the `jsconnector.xml` configuration file.

To use the Enhanced Pages page to create a new page mapping, click the **Add a Page** link and complete the necessary information for the mapping.

Page Info

The Page Info configuration options for the page mapping define the URL pattern that acts as the trigger for this mapping, plus some additional settings for the way this page mapping is handled by Interact.

Setting	Description	Equivalent setting in jsconnector.xml
URL contains	This is the URL pattern that the Web Connector should watch for in the incoming page request. For example, if the requesting URL contains "mortgage.htm" you might match that to your mortgage information page.	*<urlPattern>*
Friendly name for this page or set of pages	A meaningful name for your own reference that describes what this page mapping is for, such as "Mortgage Information Page".	*<friendlyName>*
Also return offers as JSON data for JavaScript use	A drop-down list to indicate whether you want the Web Connector to include the raw offer data in JavaScript Object Notation (http://www.json.org/) format at the end of the page content.	*<enableRawDataReturn>*

Events to fire (onload) when a visit is made to this page or set of pages

These set of configuration options for the page mapping define the URL pattern that acts as the trigger for this mapping, plus some additional settings for the way this page mapping is handled by Interact.

Note: Configuration settings in this section correspond to the `<pageLoadEvents>` section of the `jsconnector.xml`.

Setting	Description	Equivalent setting in jsconnector.xml
Individual events	A list of events that are available for this page or set of pages. The events in this list are those that you have defined in Interact, Select one or more events that you want to occur when the page is loaded. The sequence of Interact API calls is the following: 1. `startSession` 2. `postEvent` for each individual page load event (provided you have defined the individual events in Interact) 3. For each Interaction Point: • `getOffers` • `postEvent(ContactEvent)`	*<event>*

Interaction Points (offer display locations) on this page or set of pages

These set of configuration options for the page mapping allow you to select which Interaction Points appear on the pageInteract.

Note: Configuration settings in this section correspond to the `<pageMapping>` | `<page>` | `<interactionPoints>` section of the `jsconnector.xml`.

Setting	Description	Equivalent setting in jsconnector.xml
Interaction Point name checkbox	Each Interaction Point that has been defined in the configuration file appears in this section of the page. Selecting the checkbox next to the name of the Interaction Point displays a number of options available for that Interaction Point.	*<interactionPoint>*
HTML Element ID (Interact will set the innerHTML)	The name of the HTML element that should receive the content for this Interaction Point. For example, if you specified `<div id="welcomebanner">` on the page, you would enter `welcomebanner` (the ID value) in this field.	*<htmlElementId>*

Setting	Description	Equivalent setting in jsconnector.xml
HTML Display Type	A drop-down list that allows you to select the HTML Display Type (the HTML snippets, or .flt files, defined on a previous Web Connector configuration page) to use for this Interaction Point.	*<htmlSnippet>*
Maximum number of offers to present (if this is a carousel or flipbook)	The maximum number of offers that the Web Connector should retrieve from the Interact server for this Interaction Point This field is optional, and applies only for an Interaction Point that regularly updates the offers presented without reloading the page, as in the carousel scenario where multiple offers are retrieved so that they can be made available one at a time.	*<maxNumberOfOffers>*
Event to fire when the offer is presented	The name of the contact event to be posted for this Interaction Point.	*<contactEvent>*
Event to fire when the offer is accepted	The name of the accept event to be posted for this Interaction Point at the time that the offer link is clicked.	*<acceptEvent>*
Event to fire when the offer is rejected	The name of the reject event to be posted for this Interaction Point. **Note:** At this time, this feature is not yet used	*<rejectEvent>*

Web Connector Configuration Options

In general, you can use a graphical Web Connector interface to configure your Web Connector settings. All of the settings you specify are also stored in a file called jsconnector.xml, found in your `jsconnector/conf` directory. Each of the parameters that are saved in the jsconnector.xml configuration file is described here.

Parameters and their descriptions

The following parameters are stored in the `jsconnector.xml` file and are used for Web Connector interactions. There are two ways to modify these settings:

- Using the Web Connector Configuration web page that is automatically available after you have deployed and started the Web Connector application. To use the Configuration web page, use your web browser to open a URL similar to the following: `http://<host>:<port>/interact/jsp/WebConnector.jsp`.

  The changes you make on the Administration web page are stored in the `jsconnector.xml` file on the server where the Web Connector is deployed.
- Edit the `jsconnector.xml` file directly using any text editor or XML editor. Be sure that you are comfortable editing XML tags and values before using this method.

  Note: Any time you edit the `jsconnector.xml` file manually, you can reload those settings by opening the Web Connector Administration Page (found at `http://<host>:<port>/interact/jsp/jsconnector.jsp`) and clicking **Reload Configuration**.

The following table describes the configuration options you can set as they appear in the jsconnector.xml file.

Table 34. Web Connector configuration options

Parameter Group	Parameter	Description
defaultPageBehavior		
	friendlyName	A human-readable identifier for the URL Pattern for display on the Web Connector's web configuration page.
	interactURL	The base URL of the Interact runtime server. Note: You need to set this parameter only if the Web Connector (jsconnector) service is running as a deployed web application. You do not need to set this parameter if the WebConnector is running automatically as part of the Interact runtime server.
	jsConnectorURL	The base URL used to generate the click-through URL, such as http://host:port/jsconnector/clickThru
	interactiveChannel	Name of the interactive channel that represents this page mapping.
	sessionIdCookie	Name of the cookie that contains the session ID that is used in the API calls to Interact.
	visitorIdCookie	Name of the cookie to contain the audience ID.
	audienceLevel	The campaign audience level for the inbound visitor, used in the API call to the Interact runtime.
	audienceIdField	Name of the audienceId field used in the API call to the Interact runtime. **Note:** Note: There is currently no support for multi-field audience identifiers.
	audienceIdFieldType	The datatype of the audience ID field [numeric | string] used in the API call to the Interact runtime
	audienceLevelCookie	Name of the cookie that to contain the audience level. This is optional. If you do not set this parameter, the system uses what is defined for audienceLevel.
	relyOnExistingSession	Used in the API call to the Interact runtime. In general, this parameter is set to "true".
	enableInteractAPIDebug	Used in the API call to the Interact runtime to enable debugging output to the log files.
	pageLoadEvents	The event that will be posted once this particular page is loaded. Specify one or more events within this tag, in the format similar to <event>event1</event>.
	interactionPointValues	All items in this category act as default values for missing values in the IP specific categories.
	interactionPointValuescontactEvent	Default name of contact event to be posted for this particular interaction point.

Table 34. Web Connector configuration options (continued)

Parameter Group	Parameter	Description
	interactionPointValuesacceptEvent	Default name of accept event to be posted for this particular interaction point.
	interactionPointValuesrejectEvent	Default name of the reject event to be posted for this particular interaction point. (Note: at this time, this feature is not used.)
	interactionPointValueshtmlSnippet	Default name of HTML template to be served for this interaction point.
	interactionPointValuesmaxNumberOfOffers	Default max number of offers to be retrieved from Interact for this interaction point.
	interactionPointValueshtmlElementId	Default name of HTML element to receive the content for this interaction point.
	interactionPoints	This category contains the configuration for each interaction point. For any missing properties the system will rely on what's configured under the interactionPointValues category.
	interactionPointname	Name of the Interaction Point (IP).
	interactionPointcontactEvent	Name of contact event to be posted for this particular IP.
	interactionPointacceptEvent	Name of accept event to be posted for this particular IP.
	interactionPointrejectEvent	Name of the reject event to be posted for this particular IP. (Note that this feature is not yet in use.)
	interactionPointhtmlSnippet	Name of the HTML template to be served for this IP.
	interactionPointmaxNumberOfOffers	Max number of offers to be retrieved from Interactfor this IP
	interactionPointhtmlElementId	Name of the HTML element to receive the content for this interaction point.
	enableDebugMode	Boolean flag (acceptable values: true or false) to turn on special debug mode. If you set this to true, the content returned from the Web Connector includes a JavaScript call to 'alert' informing the client of the particular page mapping that just occurred. The client must have an entry in the authorizedDebugClients file to generate the alert.
	authorizedDebugClients	A file used by the special debug mode that contains the list of host names or Internet Protocol (IP) addresses that qualify for debug mode.
	enableRawDataReturn	A Boolean flag (acceptable values: true or false) to determine whether the Web Connector attaches the raw offer data in JSON format at the tail end of the content.

Table 34. Web Connector configuration options (continued)

Parameter Group	Parameter	Description
	`enableNetInsightTagging`	A Boolean flag (acceptable values: `true` or `false`) to determine whether the Web Connector attaches a Digital Analytics for On Premises tag at the end of the content.
	`apiSequence`	Represents an implementation of the APISequence interface, which dictates the sequence of API calls made by the Web Connector when a `pageTag` is called. By default, the implementation uses a sequence of `StartSession`, `pageLoadEvents`, `getOffers`, and `logContact`, where the last two are specific to each Interaction Point.
	`clickThruApiSequence`	Represents an implementation of the APISequence interface, which dictates the sequence of API calls made by the Web Connector when a `clickThru` is called. By default, the implementation uses a sequence of `StartSession` and `logAccept`.
	`netInsightTag`	Represents the HTML and JavaScript template used to integrate a call to the Digital Analytics for On Premises tag. In general, you should not need to change this option.

Using the Web Connector Admin Page

The Web Connector includes an administration page that provides some tools to help manage and test the configuration as it might be used with specific URL patterns. You can also use the Admin Page to reload a configuration that you have modified.

About the Admin Page

Using any supported web browser, you can open `http://`*`host`*`:`*`port`*`/interact/jsp/jsconnector.jsp`, where *host:port* is the host name on which the Web Connector is running and the port on which it is listening for connections, such as `runtime.example.com:7001`

You can use the Admin Page in any of the following ways:

Table 35. Web Connector Admin Page Options

Option	Purpose
Reload Configuration	Click the **Reload Configuration** link to reload any configuration changes that have been saved on disk into memory. This is necessary when you have made changes directly to the Web Connector jsconnector.xml configuration file rather than using the configuration web pages.

Table 35. Web Connector Admin Page Options (continued)

Option	Purpose
View Config	View the WebConnector configuration based on the URL pattern you enter into the **View Config** field. When you enter the URL of a page and click **View Config**, the Web Connector returns the configuration that the system will use based on that that pattern mapping. If no match is found, the default configuration is returned. This is useful for testing whether the correct configuration is being used for a particular page.
Execute Page Tag	Completing the fields on this page and clicking **Execute Page Tag** causes the Web Connector to return the `pageTag` result based on the URL pattern. This simulates the calling of a page tag. The difference between calling the pageTag from this tool and using a real web site is that using this Admin Page will cause any errors or exceptions to be displayed. For a real website, exceptions are not returned and are visible only in the Web Connector log file.

Sample Web Connector Page

As an example, a file called `WebConnectorTestPageSA.html` has been included with the Interact Web Connector (in the directory `<Interact_Home/jsconnector/webapp/html`) that demonstrates how many of the features of the Web Connector would be tagged in a page. For convenience, that sample page is also shown here.

Sample Web Connector HTML Page

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
   <meta http-equiv="Content-Type" content="text/html; charset=us-ascii" />
   <meta http-equiv="CACHE-CONTROL" content="NO-CACHE" />
<script language="javascript" type="text/javascript">
//<![CDATA[
/* ###################################################################
This is a test page that  contains the WebConnector pageTag.  Because the
name of this file has TestPage embedded, the WebConnector will detect a URL
pattern match to the url pattern "testpage" in the default version of the
jsconnector.xml - the configuration definition mapped to that "testpage"
URL pattern will apply here. That means there should this page the
corresponding html element ids that correspond to the IPs for this URL
pattern (ie. 'welcomebanner', 'crosssellcarousel', and 'textservicemessage')
################################################################### */

/* ###################################################################
This section sets the cookies for sessionId and visitorId.
Note that in a real production website, this is done most likely by the login
component. For the sake of testing, it's done here...  the name of the cookie
has to match what's configured in the jsconnector xml.
################################################################### */
     function setCookie(c_name,value,expiredays)
     {
       var exdate=new Date();
       exdate.setDate(exdate.getDate()+expiredays);
       document.cookie=c_name+ "=" +escape(value)+
       ((expiredays==null) ? "" : ";expires="+exdate.toGMTString());
     }
     setCookie("SessionID","123");
     setCookie("CustomerID","1");
```

```
   /* ###################################################
   Now set up the html element IDs that correspond to the IPs
   ################################################### */
     document.writeln("<div id='welcomebanner'> This should change, "
  + "otherwise something is wrong <\/div>");
     document.writeln("<div id='crosssellcarousel'> This should change, "
  + "otherwise something is wrong <\/div>");
     document.writeln("<div id='textservicemessage'> This should change, "
  + "otherwise something is wrong <\/div>");
   //]]&gt;
</script><!--
################################################################################
this is what is pasted from the pageTag.txt file in the conf directory of
the WebConnector installation... the var unicaWebConnectorBaseURL needs to be
tweaked to conform to your local WebConnector environment
################################################################################
-->
<!-- BEGIN: IBM Interact Web Connector Page Tag -->
<!--
# ******************************************************************************
#  Licensed Materials - Property of IBM
#  IBM Interact
#  (c) Copyright IBM Corporation 2001, 2012.
#  US Government Users Restricted Rights - Use, duplication or disclosure
#  restricted by GSA ADP Schedule Contract with IBM Corp.
# ******************************************************************************
-->
<script language="javascript" type="text/javascript">
//<![CDATA[
   var unicaWebConnectorBaseURL=
        "[CHANGE ME - http://host:port/<jsconnector>/pageTag]";
   var unicaURLData = "ok=Y";
   try {
    unicaURLData += "&url=" + escape(location.href)
   } catch (err) {}
   try {
    unicaURLData += "&title=" + escape(document.title)
   } catch (err) {}
   try {
    unicaURLData += "&referrer=" + escape(document.referrer)
   } catch (err) {}
   try {
    unicaURLData += "&cookie=" + escape(document.cookie)
   } catch (err) {}
   try {
    unicaURLData += "&browser=" + escape(navigator.userAgent)
   } catch (err) {}
   try {
    unicaURLData += "&screensize=" +
    escape(screen.width + "x" + screen.height)
   } catch (err) {}
   try {
    if (affiliateSitesForUnicaTag) {
        var unica_asv = "";
        document.write("<style id=\"unica_asht1\" type=\"text/css\"> "
     + "p#unica_ashtp a {border:1px #000000 solid; height:100px "
     + "!important;width:100px "
     + "!important; display:block !important; overflow:hidden "
     + "!important;} p#unica_ashtp a:visited {height:999px !important;"
     + "width:999px !important;} <\/style>");
        var unica_ase = document.getElementById("unica_asht1");
        for (var unica_as in affiliateSitesForUnicaTag) {
         var unica_asArr = affiliateSitesForUnicaTag[unica_as];
         var unica_ashbv = false;
         for (var unica_asIndex = 0; unica_asIndex <
     unica_asArr.length && unica_ashbv == false;
     unica_asIndex++)
```

```
   {
          var unica_asURL = unica_asArr[unica_asIndex];
          document.write("<p id=\"unica_ashtp\" style=\"position:absolute; "
     + "top:0;left:-10000px;height:20px;width:20px;overflow:hidden; \
   margin:0;padding:0;visibility:visible;\"> \
   <a href=\"" + unica_asURL + "\">" + unica_as + " <\/a><\/p>");
          var unica_ae = document.getElementById("unica_ashtp").childNodes[0];
          if (unica_ae.currentStyle) {
           if (parseFloat(unica_ae.currentStyle["width"]) > 900)
       unica_ashbv = true
          } else if (window.getComputedStyle) {
           if (parseFloat(document.defaultView.getComputedStyle
       (unica_ae, null).getPropertyValue("width")) > 900)
    unica_ashbv = true
          }
          unica_ae.parentNode.parentNode.removeChild(unica_ae.parentNode)
         }
         if (unica_ashbv == true) {
          unica_asv += (unica_asv == "" ? "" : ";") + unica_as
         }
        }
        unica_ase.parentNode.removeChild(unica_ase);
        unicaURLData += "&affiliates=" + escape(unica_asv)
    }
   } catch (err) {}
   document.write("<script language='javascript' "
     + " type='text/javascript' src='" + unicaWebConnectorBaseURL + "?"
  + unicaURLData + "'><\/script>");
  //]]&gt;
</script>
<style type="text/css">
/*<![CDATA[*/
   .unicainteractoffer {display:none !important;}
   /*]]&gt;*/
</style>
   <title>Sample Interact Web Connector Page</title>
   </head>
   <body>
   <!-- END: IBM Interact Web Connector Page Tag -->
   <!--
################################################################################
end of pageTag paste
################################################################################
-->
   </body>
</html>
```

Chapter 17. Interact and Digital Recommendations integration

IBM Interact can integrate with IBM Digital Recommendations to provide Interact-driven product recommendations. Both products can provide product recommendations for offers, but using different methods. Digital Recommendations uses a visitor's web behavior (collaborative filter) to build correlations between visitors and recommended offers. Interact is based on customer's past behavior, attributes, history, and less on view-level offers, learning which offers best match a customer's behavior profile (based on demographics and other information about the customer). Offer acceptance rates help to build a predictive model through self-learning. Using the best of both products, Interact can use a personal profile to define offers that will pass a category ID to Digital Recommendations and retrieve recommended products based on popularity (the "wisdom of the crowds") for display to the visitor as part of the selected offers. This can provide better recommendations for customers that will result in more click-throughs and better outcomes than either product acting alone.

The following sections describe how this integration works, and how to use the sample application provided to create your own custom offer integration.

Overview of Interact integration with Digital Recommendations

This section describes how IBM Interact can integrate with IBM Digital Recommendations to provide Interact-driven product recommendations, including a description of the process, and the mechanisms by which the integration takes place.

IBM Interact integrates with IBM Digital Recommendations via a Representational state transfer (REST) application programming interface (API), made available from the Digital Recommendations installation. By making the REST API calls with the appropriate category ID, Interact can retrieve recommended products and include them in the offer information displayed on the customized page that the visitor is viewing.

When a visitor views the URL of the web page (such as the sample JSP page included with your Interact installation), the page calls Interact to fetch an offer. Assuming the offer has been configured within Interact with the correct parameters, the following steps occur, in the simplest case:

1. The page logic identifies the Customer ID of the visitor.
2. An API call to Interact is made, passing in the required information to generate an offer for that customer.
3. The returned offer provides the web page with at least three attributes: the URL for the offer image, the URL of the landing page when the customer clicks through, and the category ID to use for determining which products to recommend.
4. The category ID is then used to call Digital Recommendations to retrieve the recommended products. This set of products is in JSON (JavaScript Object Notation) format in order by top-selling products in that category.
5. The offer and products are then displayed in the visitor's browser.

This integration is useful for combining offer recommendation and product recommendations together. For example, on one web page you might have two interaction points: one for an offer, and one for recommendations matching that offer. To accomplish this, the web page makes a call to Interact to make a real-time segmentation to determine best offer (say, for 10% off all small appliances). When the page receives the offer from Interact, that offer would contain the category ID (in this example, for small appliances). The page would then pass the category ID for small appliances to Digital Recommendations using an API call, and receive in response the best product recommendations for that category based on popularity.

A simpler example might be where a web page makes a call to Interact from only to find out a category (say, high-end cutlery) that matches the customer profile. It would then pass the received category ID to Digital Recommendations, and get cutlery product recommendations.

Integration Prerequisites

Before you can use the Digital Recommendations - Interact integration, you must make sure that you meet the prerequisites described in this section.

Be sure that the following prerequisites are true:

- You are familiar with the use of the Interact API as documented elsewhere in *Administrator's Guide* and online help.
- You are familiar with the Digital Recommendations REST API as described in your Digital Recommendations developer documentation.
- You have a basic understanding of HTML, JavaScript, CSS, and JSON (JavaScript Object Notation).

  JSON is important because the Digital Recommendations REST API returns the product information you request in as JSON-formatted data.
- You are familiar with server-side coding of web pages, because the demonstration application provided with Interact uses JSP (although JSP is not required).
- You have a valid Digital Recommendations account and the list of category IDs you plan to have Interact to retrieve product recommendations (the top-selling or most popular products in the category you specify).
- You have the Digital Recommendations REST API link (a URL for your Digital Recommendations environment).

  See the sample application included with yourInteract installation for an example, or see the sample code in "Using the Integration Sample Project" on page 327 for more information.

Configuring an offer for Digital Recommendations integration

Before your web page can call Digital Analytics Digital Recommendations to retrieve a recommended product, you must first configure the IBM Interact offer with the necessary information to pass to Digital Recommendations.

To set up an offer to link to Digital Recommendations, make sure the following conditions are in place first:

- Make sure that your Interact runtime server is set up and running correctly.
- Ensure that the runtime server can establish a connection with the Digital Recommendations server, including making sure that your firewall does not prevent the outgoing establishment of a standard web connection (port 80).

To set up an offer for integration with Digital Recommendations, perform the following steps.

1. Create or edit an offer for Interact.

   For information on creating and modifying offers, see the *IBM Interact User's Guide*, and the IBM Campaign documentation.
2. In addition to the other settings in the offer, make sure that the offer includes the following offer attributes:
   - The URL (uniform resource locator) that links to the image for the offer.
   - The URL that links to the landing page for the offer.
   - An Digital Recommendations category ID associated with this offer.

     You can retrieve the category ID manually from your Digital Recommendations configuration. Interact cannot retrieve category ID values directly.

   In the demonstration web application included with your Interact installation, these offer attributes are called `ImageURL`, `ClickThruURL`, and `CategoryID`. The names can be any that are meaningful to you, as long as your web application matches the values that the offer is expecting.

   For example, you might define an offer called "10PercentOff" that contains these attributes, where the Category ID (as retrieved from your Digital Recommendations configuration) is `PROD1161127`, the URL of the offer click-through is `http://www.example.com/success`, and the URL of the image to display for the offer is `http://localhost:7001/sampleIO/img/10PercentOffer.jpg` (a URL that is, in this case, local to the Interact runtime server).
3. Define the treatment rules for an interactive channel to include this offer, and deploy the interactive channel as usual.

The offer is now defined with the information required for Digital Recommendations integration. The remaining work to allow Digital Recommendations to provide product recommendations to Interact is accomplished by configuring your web pages to make the appropriate API calls.

When you configure your web application to serve the integrated page to visitors, make sure that the following files are included in the `WEB-INF/lib` directory:

- *`Interact_Home`*`/lib/interact_client.jar`, required to handle calls from your web page to the Interact API.
- *`Interact_Home`*`/lib/JSON4J_Apache.jar`, required to handle the data returned from the call to the Digital Recommendations REST API, which returns JSON-formatted data.

See "Using the Integration Sample Project" for more information on how to serve the offers to your customers.

Using the Integration Sample Project

Every Interact run time installation includes a sample project that demonstrates the Digital Recommendations - Interact integration process. The sample project provides a complete, end-to-end demonstration of creating a web page that calls an offer that contains a category ID, which is then passed to Digital Recommendations to retrieve a recommended product list for presentation in the interaction points of the page.

Overview

You can use the included sample project as it is provided, if you want to test the integration process, or you can use it as a starting point to develop your own custom pages. The sample project is found in the following file:

Interact_home/samples/IntelligentOfferIntegration/MySampleStore.jsp

This file, in addition to containing a full, working example of the integration process, also contains extensive comments that explain what to set up in Interact, what to customize in the .jsp file, and how to deploy the page properly to run with your installation.

MySampleStore.jsp

For convenience, the MySampleStore.jsp file is shown here. This sample may be updated with subsequent releases of Interact, so use the file included with your installation as a starting point for any examples you need.

```
<!--
# ****************************************************************************
#  Licensed Materials - Property of IBM
#  IBM Interact
#  (c) Copyright IBM Corporation 2001, 2011.
#  US Government Users Restricted Rights - Use, duplication or disclosure
#  restricted by GSA ADP Schedule Contract with IBM Corp.
# ****************************************************************************

-->


<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ page import="java.net.URL,
    java.net.URLConnection,
    java.io.InputStreamReader,
    java.io.BufferedReader,
    com.unicacorp.interact.api.*,
    com.unicacorp.interact.api.jsoverhttp.*,
    org.apache.commons.json.JSONObject,
    org.apache.commons.json.JSONArray" %>

<%

 /*****************************************************************************
 * This sample jsp program demonstrates integration of Interact and Digital Recommendations.
 *
 * When the URL for this jsp is accessed via a browser. the logic will call Interact
 * to fetch an Offer.  Based on the categoryID associated to the offer, the logic
 * will call Digital Recommendations to fetch recommended products.  The offer and products
 * will be displayed.
 * To toggle the customerId in order to demonstrate different offers, one can simply
 * append cid=<id> to the URL of this JSP.
 *
 * Prerequisites to understand this demo:
 * 1) familiarity of Interact and its java API
 * 2) familiarity of IntelligentOffer and its RestAPI
 * 3) some basic web background ( html, css, javascript) to mark up a web page
 * 4) Technology used to generate a web page (for this demo, we use JSP executed on the server side)
 *
 *
 * Steps to get this demo to work:
 * 1) set up an Interact runtime environment that can serve up offers with the following
 * offer attributes:
 *  ImageURL : url that links to the image of the offer
 *  ClickThruURL : url that links to the landing page of the offer
 *  CategoryID : Digital Recommendations category id associated to the offer
 * NOTE: alternate names for the attributes may be used as long as the references to those
 * attributes in this jsp are modified to match.
 * 2) Obtain a valid REST API URL to the Intelligent Offer environment
 * 3) Embed this JSP within a Java web application
 * 4) Make sure interact_client.jar is in the WEB-INF/lib directory (communication with Interact)
 * 5) Make sure JSON4J_Apache.jar (from interact install) is in the
```

```
 *    WEB-INF/lib directory (communication with IO)
* 6) set the environment specific properties in the next two sections
*****************************************************************************/

/*****************************************************************************
*   ****************CHANGE THESE SETTINGS TO REFLECT YOUR ENV*******************
* Set your Interact environment specific properties here...
*****************************************************************************/

final String sessionId="123";
final String interactiveChannel = "SampleIO";
final String audienceLevel = "Customer";
final String audienceColumnName="CustomerID";
final String ip="ip1";
int customerId=1;
final String interactURL="http://localhost:7011/interact/servlet/InteractJSService";
final boolean debug=true;
final boolean relyOnExistingSession=true;

/*****************************************************************************
  ****************CHANGE THESE SETTINGS TO REFLECT YOUR ENV*******************
* Set your Digital Recommendations environment specific properties here...
*****************************************************************************/

final String ioURL="http://recs.coremetrics.com/iorequest/restapi";
final String zoneID="ProdRZ1";
final String cID="90007517";



/*****************************************************************************
  *****************************************************************************/

StringBuilder interactErrorMsg = new StringBuilder();
StringBuilder intelligentOfferErrorMsg = new StringBuilder();

// get the customerID if passed in as a parameter
String cid = request.getParameter("cid");
if(cid != null)
{
 customerId = Integer.parseInt(cid);
}

// call Interact to get offer
Offer offer=getInteractOffer(interactURL,sessionId,interactiveChannel,audienceLevel,
       audienceColumnName,ip,customerId,debug,relyOnExistingSession,interactErrorMsg);

// get specific attributes from the offer (img url, clickthru url, & category id)
String offerImgURL=null;
String offerClickThru=null;
String categoryId="";

if(offer != null)
{
 for(NameValuePair offerAttribute : offer.getAdditionalAttributes())
    {
        if(offerAttribute.getName().equalsIgnoreCase("ImageURL"))
        {
         offerImgURL=offerAttribute.getValueAsString();
        }
        else if(offerAttribute.getName().equalsIgnoreCase("ClickThruURL"))
        {
         offerClickThru=offerAttribute.getValueAsString();
        }
        else if(offerAttribute.getName().equalsIgnoreCase("CategoryID"))
        {
         categoryId=offerAttribute.getValueAsString();
        }
    }
}

// call Digital Recommendations to get products
JSONObject products=getProductsFromIntelligentOffer(ioURL, cID, zoneID, categoryId,
       intelligentOfferErrorMsg);

%>

<html>
 <head>
```

```
 <title>My Favorite Store</title>

 <script language="javascript" type="text/javascript">
  var unicacarousel=(function(){var g=false;var h;var j=0;var k=0;var l=0;var m=40;
         var n=new Array(0,2,6,20,40,60,80,88,94,97,99,100);var o=function(a){var b=a.parentNode;
         h=b.getElementsByTagName("UL")[0];var c=h.getElementsByTagName("LI");j=c[0].offsetWidth;
         k=c.length;l=Math.round((b.offsetWidth/j));unicacarousel.recenter()};var p=function(a)
         {var b=parseFloat(h.style.left);if(isNaN(b))b=0;for(var i=0;i<n.length;i++)
         {setTimeout("unicacarousel.updateposition("+(b+(a*(n[i]/100)))+");",((i*m)+50))}
         setTimeout("unicacarousel.recenter();",((i*m)+50))};return{gotonext:function(a,b)
         {if(!g){o(a);g=true;p((-1*b*j))}},gotoprev:function(a,b){if(!g){o(a);g=true;p((b*j))}},
         updateposition:function(a){h.style.left=a+"px"},recenter:function(){var a=parseFloat(h.style.left);
         if(isNaN(a))a=0;var b=j*Math.round(((l-k)/2));var c=Math.abs(Math.round((b-a)/j));
         if(a<b){var d=h.getElementsByTagName("LI");var e=new Array();
         for(var i=0;i<c;i++){e[e.length]=d[i]}for(var i=0;i<e.length;i++)
         {h.insertBefore(e[i],null)}unicacarousel.updateposition(b)}else
         if(a>b){var d=h.getElementsByTagName("LI");var e=new Array();
         for(var i=0;i<c;i++){e[e.length]=d[d.length-c+i]}var f=d[0];
         for(var i=0;i<e.length;i++){h.insertBefore(e[i],f)}unicacarousel.updateposition(b)}g=false}}})();
 </script>

 <style type="text/css">
.unicaofferblock_container {width:250px; position:relative; display:block;
                            text-decoration:none; color:#000000; cursor: pointer;}
.unicaofferblock_container .unicateaserimage {margin:0px 0.5em 0.25em 0px; float:left;}
.unicaofferblock_container .unicabackgroundimage {position:absolute; top:0px; left:0px;}
.unicaofferblock_container .unicabackgroundimagecontent {width:360px; height:108px;
                    padding:58px 4px 4px 20px; position:relative; top:0px;}
.unicaofferblock_container h4 {margin:0px; padding:0px; font-size:14px;}

.unicacarousel {width:588px; position:relative; top:0px;}
.unicacarousel_sizer {width:522px; height:349px; margin:0px 33px; padding:0;
                      overflow:hidden; position:relative;}
.unicacarousel_rotater {height:348px; width:1000px; margin:0 !important;
                        padding:0; list-style:none; position:absolute; top:0px;
                        left:0px;}
.unicacarousel li {width:167px; height:349px; float:left; padding:0 4px;
                   margin:0px !important; list-style:none !important;
                   text-indent:0px !important;}
.unicacarousel_gotoprev, .unicacarousel_gotonext {width:18px; height:61px;
                    top:43px; background:url(../img/carouselarrows.png) no-repeat;
                   position:absolute; z-index:2; text-align:center; cursor:pointer;
                   display:block; overflow:hidden; text-indent:-9999px;
                   font-size:0px; margin:0px !important;}
.unicacarousel_gotoprev {background-position:0px 0; left:0;}
.unicacarousel_gotonext {background-position:-18px 0; right:0;}

 </style>


</head>

<body>

 <b>Welcome To My Store</b> Mr/Mrs. <%=customerId %>
 <br><br>
<% if(offer != null) { %>
 <!-- Interact Offer HTML -->

 <div onclick="location.href='<%=offerClickThru %>'" class="unicaofferblock_container">
  <div class="unicabackgroundimage">
   <a href="<%=offerClickThru %>"><img src="<%=offerImgURL %>" height="170"
         width="695" border="0"></a>
  </div>
 </div>


<% } else { %>
 No offer available.. <br> <br>
 <%=interactErrorMsg.toString() %>
<% } %>

<% if(products != null) { %>
 <!-- IntelligentOffer Products HTML -->
 <br><br><br> <br><br><br> <br><br><br> <br><br><br> <br>
 <div class="unicacarousel">
 <div class="unicacarousel_sizer">
  <ul class="unicacarousel_rotater">
```

```
 <% JSONArray recs = products.getJSONObject("io").getJSONArray("recs");
  if(recs != null)
  {
   for(int x=0;x< recs.length();x++)
   {
    JSONObject rec = recs.getJSONObject(x);
    if(rec.getString("Product Page") != null &&
               rec.getString("Product Page").trim().length()>0) {
    %>

    <li>
     <a href="<%=rec.getString("Product Page") %>" title="<%=rec.getString("Product Name") %>">
             <img src="<%=rec.getString("Image") %>"  width="166" height="148" border="0" />
     <%=rec.getString("Product Name") %>
     </a>
    </li>

    <% }
   }
  }
 %>
 </ul>
 </div>
 <p class="unicacarousel_gotoprev" onclick="unicacarousel.gotoprev(this,1);"></p>
 <p class="unicacarousel_gotonext" onclick="unicacarousel.gotonext(this,1);"></p>
</div>
<% } else { %>
 <div>
 <br><br> <br><br><br> <br><br><br> <br><br><br> <br>
 No products available...<br> <br>
 <%=intelligentOfferErrorMsg.toString() %>
 </div>
<% } %>

 </body>
</html>



<%!
 /*****************************************************************************
 * The following are convenience functions that will fetch from Interact and
 *   Digital Recommendations
 *****************************************************************************/

 /*****************************************************************************
 * Call Digital Recommendations to retrieve recommended products
 *****************************************************************************/
 private JSONObject getProductsFromIntelligentOffer(String ioURL, String cID,
     String zoneID, String categoryID, StringBuilder intelligentOfferErrorMsg)
 {

  try
  {

   ioURL += "?cm_cid="+cID+"&cm_zoneid="+zoneID+"&cm_targetid="+categoryID;
   System.out.println("CoreMetrics URL:"+ioURL);
   URL url = new java.net.URL(ioURL);

    URLConnection conn = url.openConnection();

    InputStreamReader inReader = new InputStreamReader(conn.getInputStream());
    BufferedReader in = new BufferedReader(inReader);

    StringBuilder response = new StringBuilder();

    while(in.ready())
    {
     response.append(in.readLine());
    }

    in.close();

    intelligentOfferErrorMsg.append(response.toString());

    System.out.println("CoreMetrics:"+response.toString());

    if(response.length()==0)
```

```
    return null;

   return new JSONObject(response.toString());
 }
 catch(Exception e)
 {
  intelligentOfferErrorMsg.append(e.getMessage());
  e.printStackTrace();
 }

 return null;

}

/*******************************************************************************
* Call Interact to retrieve offer
*******************************************************************************/
private Offer getInteractOffer(String interactURL,String sessionId,String interactiveChannel,
    String audienceLevel,
       String audienceColumnName,String ip, int customerId,boolean debug,
                    boolean relyOnExistingSession, StringBuilder interactErrorMsg)
{
 try
 {
  InteractAPI api = InteractAPI.getInstance(interactURL);
  NameValuePairImpl custId = new NameValuePairImpl();
     custId.setName(audienceColumnName);
      custId.setValueAsNumeric(Double.valueOf(customerId));
    custId.setValueDataType(NameValuePair.DATA_TYPE_NUMERIC);
     NameValuePairImpl[] audienceId = { custId };

      // call startSession
     Response response = api.startSession(sessionId, relyOnExistingSession,
         debug, interactiveChannel, audienceId, audienceLevel, null);

     if(response.getStatusCode() == Response.STATUS_ERROR)
     {
      printDetailMessageOfWarningOrError("startSession",response, interactErrorMsg);
  }

     // call getOffers
     response = api.getOffers(sessionId, ip, 1);
     if(response == null  || response.getStatusCode() == Response.STATUS_ERROR)
     {
      printDetailMessageOfWarningOrError("getOffers",response, interactErrorMsg);
  }

           OfferList offerList=response.getOfferList();

           if(offerList != null && offerList.getRecommendedOffers() != null)
           {
            return offerList.getRecommendedOffers()[0];
           }
 }
 catch(Exception e)
 {
  interactErrorMsg.append(e.getMessage());
  e.printStackTrace();
 }
 return null;
}

   private void printDetailMessageOfWarningOrError(String command, Response response,
       StringBuilder interactErrorMsg)
   {
    StringBuilder sb = new StringBuilder();
       sb.append("Calling "+command).append("<br>");
       AdvisoryMessage[] messages = response.getAdvisoryMessages();

       for(AdvisoryMessage msg : messages)
       {
        sb.append(msg.getMessage()).append(":");
        sb.append(msg.getDetailMessage());
        sb.append("<br>");
       }
       interactErrorMsg.append(sb.toString());
   }
%>
```

Chapter 18. Interact and Digital Data Exchange integration

With Digital Data Exchange, your website can link to Interact to provide a powerful omni-channel execution engine that delivers the best offers to the optimum channels and evolves (learns) from the offer feedback to continuously increase marketing effectiveness.

You can use this tool if your marketing team uses Interact for omni-channel offer management and wants to extend these personalized intelligent offers to your websites.

IBM Digital Data Exchange integrates IBM and third party marketing solutions with digital customer insights through a real-time data syndication API and an enterprise-grade tag management solution.

Without IBM Digital Data Exchange, marketers depend on IT to link Interact to their website and call theInteract API from various webpages. With IBM Digital Data Exchange, marketers can bypass IT and go directly to IBM Digital Data Exchange to include IBM Digital Data Exchange tags on various webpages.

Prerequisites

Before you can use the Interact and Digital Data Exchange integration, you must make sure that you meet the prerequisites described in this section.

Be sure that the following prerequisites are true.

- You are familiar with the Interact JavaScript API as documented elsewhere in Administrator's Guide and online help.
- You are familiar with the Digital Data Exchange tagging and page groups.
- You have a valid Digital Data Exchange account.
- Your `interactapi.js` file is publicly hosted so it can be accessed in **Vendor** settings.

Integrating IBM Interact with your website through IBM Digital Data Exchange

Use these steps to integrate Interact with your website through Digital Data Exchange.

1. Specify the location of the `Interactapi.js` file.
   a. Navigate to **Vendors** > **Vendor Settings** in Digital Data Exchange.
   b. Select IBM Interact from the **Vendor** drop-down.
   c. In **Library Path**, enter the URL where you hosted the `Interactapi.js`. Do not include the protocol (http or https) in this URL.
   d. In **Path To Public Rest Servlet**, add the path to the Rest Servlet.
2. Navigate to **Manage** > **Global Settings** in Digital Data Exchange to specify the object name to use as the page identifier in **Unique Page Identifier**. For example, you can set the object name to `digitalData.pageInstanceID`.

3. Include the `eluminate.js` file and an identifier on the web page where you want Digital Data Exchange to insert the tags. You should give each web page a unique identifier so Digital Data Exchange can distinguish between various pages.

   For example, you can add the following script to your home page.

```
<!-- Setting Page Identifier -->
          <script>
                  digitalData={pageInstanceID:"INTERACT_HomePage"};
          </script>

<!-- Including eluminate script -->
         <script type="text/javascript" src="http://libs.
                  coremetrics.com/eluminate.js">
         </script>
         <script type="text/javascript">
                  cmSetClientID("51310000|INTERACTTEST",false,"data.
                  coremetrics.com",document.domain);
         </script>
```

4. In Digital Data Exchange create tags, code segments, functions, and other items you want to add to your web page.
5. Create page groups to define what you want filed on each page.

   See the IBM Digital Data Exchange User Guide for more information.

Interact tags in Digital Data Exchange

Use the default Digital Data Exchange tags to define variations of the tags that are appropriate to web pages where data is represented from different locations. Once defined, these tags are added to the Interact tag list. Tags may not have fields to define or may not have required tag fields and can be used directly.

The following Interact tags are available in Digital Data Exchange under **Tags**.

- End Session
- Get Offers
- Load Library
- Post Event
- Set Audience
- Start Session

To use the Interact tags, edit the tags to define the Tag Field, Method, Object Name, Data Type, and Modifier for each Interact tag.

The Post Event, Set Audience, and Start Sessions tags accept custom tag fields. Use the Tag Field Add icon, the click the Edit icon to define the custom parameter. The process is the same as any parameter definition with the exception that the name of the parameter can be edited and must include the parameter name, a colon, and the parameter data type. Custom parameter order in the tag can be modified with the up and down arrows.

Tags can also be bound to JavaScript functions or HTML objects so that they fire after the function fires or on an HTML object event.

For more information on how to define, bind, and work with tags, see the IBM Digital Data Exchange User Guide.

For detailed use cases of the Interact and Digital Data Exchange integration, see https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W214f7731a379_4712_a1ce_5d7a833d4cca/page/IBM%20Interact%20and%20IBM%20Digital%20Data%20Exchange%20Integration.

End Session

The End Session tag marks the end of a web session.

The following tag fields are available for the End Session tag.

Table 36. End Session tags

Tag Field	Description
*Session ID	Identifies the Session ID.
On Success Callback Function Name	Defines the name of the function to be called when the end session method is successful.
On Failure Callback Function Name	Defines the name of the function to be called when the end session method fails.

Any **Tag Field** marked with an * is required.

Get Offers

Use the Get Offers tag to request offers from the runtime server.

The following tag fields are available for the Get Offers tag.

Table 37. Get Offers tags

Tag Field	Description
*Session ID	Identifies the Session ID.
*Interact Point Name	Identifies the name of the interaction point this method references. This name must match the name of the interaction point defined in interactive channel exactly.
*Number Requested	Identifies the number of offers requested.
On Success Callback Function Name	Defines the name of the function to be called when the get offers method is successful.
On Failure Callback Function Name	Defines the name of the function to be called when the get offers method fails.

Any **Tag Field** marked with an * is required.

The Get Offers tag should be assigned to a page group whose container is set to `Default`.

Load Library

The Load Library tag loads the Interact JavaScript library in the head section of the page.

The Load Library tag has no parameters. It takes the library location from the `Library Path` in **Vendor Settings**. It should be included in a page group using a container set to `Head` and should run on every page that has Interact tagging.

Important: None of the other tags will work if the load library tag is not included. The interact.js is not loaded if this tag is not included.

Post Event

Use the Post Event tag to execute any event defined in the interactive channel.

The following tag fields are available for the Post Event tag.

Table 38. Post Event tags

Tag Field	Description
*Session ID	Identifies the Session ID.
*Event Name	Identifies the name of the event. The name of the event must match the name of the event as defined in the interactive channel. This name is case-insensitive.
On Success Callback Function Name	Defines the name of the function to be called when the post event method is successful.
On Failure Callback Function Name	Defines the name of the function to be called when the post event method fails.

Any **Tag Field** marked with an * is required.

Optional parameters can be added with the custom tag field feature. Custom tag names must consist of the parameter name, a colon, and the data type.

Set Audience

Use the Set Audience tag to set the audience ID and level for a visitor.

The following tag fields are available for the Set Audience tag.

Table 39. Set Audience tags

Tag Field	Description
*Session ID	Identifies the Session ID.
*Audience ID	Identifies the Audience ID. The names must match the physical column names of any table containing the Audience ID. The Audience ID cannot contain more that 17 significant digits. If an Audience ID is more than 17 significant digits must be partitioned or the Audience ID must be changed to a string.
*Audience Level	Defines the Audience Level.
On Success Callback Function Name	Defines the name of the function to be called when the set audience method is successful.
On Failure Callback Function Name	Defines the name of the function to be called when the set audience method fails.

Any **Tag Field** marked with an * is required.

Optional parameters can be added with the custom tag field feature. Custom tag names must consist of the parameter name, a colon, and the data type.

Start Session

The Start Session tag creates and defines a web session.

The following tag fields are available for the Start Session tag.

Table 40. Start Session tags

Tag Field	Description
*Session ID	Identifies the Session ID.
*Interact Channel	Defines the name of the interactive channel this session refers to. This name must match the name of the interactive channel defined in Campaign exactly.
*Audience ID	Identifies the Audience ID. The names must match the physical column names of any table containing the Audience ID.
*Audience Level	Defines the Audience Level.
*Rely on Existing Session	Defines whether this session uses a new or an existing session
*Debug	Enables or disables debug information.
On Success Callback Function Name	Defines the name of the function to be called when the start session method is successful.
On Failure Callback Function Name	Defines the name of the function to be called when the start session method fails.

Any **Tag Field** marked with an * is required.

Optional parameters can be added with the custom tag field feature. Custom tag names must consist of the parameter name, a colon, and the data type.

The Start Session tag should be assigned to a page group whose container is set to `Default`.

Example tag settings

This example shows a simple configuration of the Start Session, Post Event, Get Offers, and End Session tag settings.

For any tag, you can get the tag field values from the cookie with the cookie method or from the JavaScript object with the javascriptobject method.

These tags support additional parameters that this simple example does not show. You can find more information on the additional parameters in the IBM Digital Data Exchange User Guide.

For detailed use cases of the Interact and Digital Data Exchange integration, see https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W214f7731a379_4712_a1ce_5d7a833d4cca/page/IBM%20Interact%20and%20IBM%20Digital%20Data%20Exchange%20Integration.

Example Start Session tag settings

Click **Tags** > **IBM Tags** > **IBM Interact** > **Type: Start Session** to create a Start Session tag. Edit the tag with the following settings.

Session ID settings

- **Method**: `Constant`
- **Constant**: `5555`
- **Data Type**: `String`
- **Modifier**: `<null>`

Interactive Channel settings

- **Method**: `Constant`
- **Constant**: `WSCDemo`
- **Data Type**: `String`
- **Modifier**: `<null>`

Audience ID settings

- **Method**: `Constant`
- **Constant**: `USERS_ID,2002,numeric`
- **Data Type**: `String`
- **Modifier**: `<null>`

Audience Level settings

- **Method**: `Constant`
- **Constant**: `WSCUserId`
- **Data Type**: `String`
- **Modifier**: `<null>`

Rely On Existing Session settings

- **Method**: `Constant`
- **Constant**: `False`
- **Data Type**: `Boolean`
- **Modifier**: `<null>`

Debug

- **Method**: `Constant`
- **Constant**: `True`
- **Data Type**: `Boolean`
- **Modifier**: `<null>`

On Success Callback Function Name settings

- **Method**: `Unassigned`
- **Value**: `<null>`

On Failure Callback Function Name settings

- **Method**: `Unassigned`
- **Value**:`<null>`

Example Get Offers tag settings

Click **Tags** > **IBM Tags** > **IBM Interact** > **Type: Get Offers** to create a Get Offers tag. Edit the tag with the following settings.

Session ID settings

- **Method**: `Constant`
- **Constant**: `5555`
- **Data Type**: `String`
- **Modifier**: `<null>`

Interact Point Name settings

- **Method**: `Constant`
- **Constant**: `AuroraHomepageHeaderBannerLeft`
- **Data Type**: `String`
- **Modifier**: `<null>`

Number Requested settings

- **Method**: `Constant`
- **Constant**: `1`
- **Data Type**: `integer`
- **Modifier**: `<null>`

On Success Callback Function Name settings

- **Method**: `Constant`
- **Constant**: `onOfferReturnSuccess`
- **Data Type**: `string`
- **Modifier**: `<null>`

On Failure Callback Function Name settings

- **Method**: `Constant`
- **Constant**: `onOfferReturnError`
- **Data Type**: `string`
- **Modifier**: `<null>`

Example Post Event tag settings

Click **Tags** > **IBM Tags** > **IBM Interact** > **Type: Post Event** to create a Post Event tag. Edit the tag with the following settings.

Session ID settings

- **Method**: `Constant`
- **Constant**: `5555`
- **Data Type**: `String`
- **Modifier**: `<null>`

Event Name settings

- **Method**: `Constant`
- **Constant**: `ACCEPTOFFER`
- **Data Type**: `String`
- **Modifier**: `<null>`

On Success Callback Function Name settings

- **Method**: `Constant`

- **Constant**: onSuccessTestFunction
- **Data Type**: String
- **Modifier**: <null>

On Failure Callback Function Name settings

- **Method**: Constant
- **Constant**: onErrorTestFunction
- **Data Type**: String
- **Modifier**: <null>

Additional parameter field settings

- **Tag Field**: UACIOfferTrackingCode:string
- **Method**: JavaScriptObject
- **Object Name**: oa.treatmentCode
- **Data Type**: String
- **Modifier**: <null>

Example End Session tag settings

Click **Tags** > **IBM Tags** > **IBM Interact** > **Type: End Session** to create an End Session tag. Edit the tag with the following settings.

Session ID settings

- **Method**: Constant
- **Constant**: 5555
- **Data Type**: String
- **Modifier**: <null>

On Success Callback Function Name settings

- **Method**: Unassigned
- **Value**: <null>

On Failure Callback Function Name settings

- **Method**: Unassigned
- **Value**: <null>

Example functions

For the functions used for the On Success Callback Function Name and On Failure Callback Function Name settings, you only have to specify the function name when you create a new tag if the function is already present on your webpage.

You can also use the Digital Data Exchange Utilities to create functions and add them to your webpages.

The following example shows how to display an offer returned from Interact on your webpage. You must include this script on the page or use the Digital Data Exchange code snippet to inject it.

```
<script>
oa = {treatmentCode: ""};
function acceptOffer(treatmentCode) {
oa.treatmentCode = treatmentCode;
```

```
}
function onOfferReturnSuccess(response) {
var offer = response.offerList[0].offers[0];
var attributes = offer.attributes;
var offerText = "";
var offerLinkURL = "#";
for(var i = 0; i<attributes.length; i++)
{
if(attributes[i].n == "OfferTerms")
{
offerText = attributes[i].v;
}
else if(attributes[i].n == "OfferLinkURL")
{
offerLinkURL = attributes[i].v;
}
}

var link = "<a href=\"'+offerLinkURL+"\" onclick=\"acceptOffer
('"+offer.treatmentCode+"')\">"+offerText+"</a>";
document.getElementById("offerContainer").innerHTML="
<div style=\"text-align:center;padding:
10px 0;background-color:#f5f5f5;\">"+link+"</div>";
}
function onOfferReturnError(response) {
(JSON.stringify(response));
}
</script>
```

Verify your integration configuration

Use the Digital Data Exchange test tool and the `Interact.log` file to troubleshoot any configuration problems.

You can use the Digital Data Exchange test tool to check the encyclopedia to see if your configuration works as expected. To open the test tool, click **Deployment** > **Test Tool** in Digital Data Exchange.

See the IBM Digital Data Exchange User Guide for more information on the test tool.

You can view the `Interact.log` file to see details about the various Interact API calls that are made. Add the On Success Callback Function and On Failure Callback Function to each tag to debug the various calls.

Chapter 19. Configure gateways for triggered messages

Use triggered message gateways to send and receive offer information from inbound and outbound channels.

You can use the following inbound and outbound gateways with triggered messages.

- IBM Interact Inbound Gateway for IBM Universal Behavior Exchange
- IBM Interact Outbound Gateway for IBM Universal Behavior Exchange
- IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud
- IBM Interact Outbound Gateway for IBM Mobile Push Notification

For more information, see https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W214f7731a379_4712_a1ce_5d7a833d4cca/page/IBM%20Interact%20Triggered%20Messages.

Using the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange

To use the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange, you must configure Interact, configure a UBX subscriber endpoint, and create an endpoint and event in UBX.

Use the following configurations as an example for your configuration.

You can download the subscriber gateway from http://www.ibm.com/support/fixcentral/swg/quickorder?parent=Enterprise%2BMarketing%2BManagement&product=ibm/Other+software/Unica+Interact&release=All&platform=All&function=fixId&fixids=IBM_Interact_OMO_Gateway_for_UBX_Subscriber_2.0&includeRequisites=1&includeSupersedes=0&downloadMethod=http&source=fc.

Configuring Interact for the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange

Use the following steps to configure Interact.

1. In the **Interact | activityOrceshtrator | receivers** configuration property, add a new receiver. Set **Type** to `IBMMQ` or `Custom`. If you choose `Custom`, enter **ClassName** and **ClassPath**. If you choose `IBMMQ`, leave **ClassPath** and **ClassName** blank.
2. Add `providerURL`, `queueManager`, `messageQueueName`, `authDS`, and `asmUserFor...."` parameters for your receiver.
3. In the **Interact | activityOrceshtrator | gateways** configuration property, add a new gateway. Set **ClassPath** to the URI of the location of the `OMO_InteractGateway_UBX.jar` file and **ClassName** to

   ```
   com.ibm.interact.offerorchestration.inboundgateway.ubx.
   UBXInboundGateway
   ```
4. Create a `Interactubx11` folder under the `UBX` folder of the inbound gateway and copy the properties files to this new folder. The folder name should match the name of the subscriber endpoint that you created in UBX.

5. In the interactEventNamemapping.properties file, add an entry to map the value of the payload event field to the Interact event name. For example, recommededOffers=recommendedOffers.
6. In the interactEventPayloadMapping.properties file, add your field definitions with the names of these parameters set to OMO-conf_inbound_UBX_interactEventNameMapping and OMO-conf_inbound_UBX_interactEventNameMapping, respectively

   For example:

   ```
   [SessionID]=(String)interactprofileid
   [EventName]=(String)code
   [AudienceIDFieldNames]=(String)"CustomerID"
   [AudienceIDFieldValues]=(Numeric)interactprofileid
   [AudienceLevel]=(String)"Customer"
   [InteractChannel]=(String"UBX_MM"
   ```
7. Add the locations of your Interactubx11/interactEven tNameMapping.properties and Interactubx11/interactEven tPayloadMapping.properties as parameters for your gateway under **Interact | activityOrceshtrator | gateways | [gatewayname] | Parameter Data**.
8. Create an interactive channel and add an event to the interactive channel.
9. Add a triggered messages rule with the recommendedOffers event and assign an offer to the rule.
10. Deploy the interactive channel.
11. Restart the Interact server.
12. Post an event to UBX with a REST API client.

    Example event body:

    ```
    {
      "channel" : "mobile",
      "identifiers" : [
        {
          "name" : "interactprofileid",
          "value" : "55"
        }
      ],
    "events" : [
      {
        "code" : "recommendedOffers",
        "timestamp" : "2015-12-28T20:16:12Z"
       }
      ]
    }
    ```
13. Check the Interact log to see if the triggered messages event is triggered.

Configuring the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint

This is a sample endpoint that you can use as an example.

You should also use the instructions to complete the following configurations.

- UBX endpoint with IBM MQ
- Endpoint ubxInboundEndpoint.properties file
- Endpoint inboundProducerNameConfig.properties file
- Endpoint inboundQueueNameConfig.properties file
- Endpoint log4j.properties file

Deploying the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange and endpoint

1. Download and unzip `IBM_Interact_OMO_Gateway_for_UBX_Subscriber_2.0.zip` to the directory in which you installed Interact on the Interact runtime server.
2. Download and unzip `IBM_Interact_OMO_Endpoint_for_UBX_Subscriber_2.0.zip` to any directory (for example, c:\ubxInboundEndpoint) on a publicly accessible JavaEE enabled application server or web server. This server will post data to the Interact inbound JMS Queue to be later consumed by the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange.

Configuring IBM Interact Inbound Gateway for IBM Universal Behavior Exchange Interact Inbound Gateway endpoint

The IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint is configured to accept requests from Universal Behavior Exchange and send it to the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange.

You must complete the following tasks to configure the Universal Behavior Exchange Subscriber Gateway endpoint

1. A new Java system property (-DubxInboundEndpointConfigPath) needs to be configured by editing the configuration file in the web server or in the administrative console of application server. The -D property should point to the endpoint install directory in the server. This directory contains configuration files for the target JMS queue and various logging levels for the endpoint. For example -DubxInboundEndpointConfigPath=c:\ ubxInboundEndpoint.
2. Deploy the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint web archive file (ubxInboundEndpoint.war) from the install directory as described in the web server or application server documentation.

To verify that the endpoint was installed correctly, enter the following address into any browser and look for message `UBX End Point is UP`.

`http://[Server]:[Port]/[ContextRoot]/UBXEndPoint`

Note: You should protect the publicly accessible IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint by adding necessary firewall rules to accept http request from IBM Universal Behavior Exchange Server only.

For example, you can use the following instructions to configure and deploy IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint on WebSphere Application Server.

1. Open the administrative console.
2. Select **Servers** > **(Expand Server Types)** > **server_name** > **(Expand Java™ and Process Management)** > **Process Definition** > **Java Virtual Machine**.
3. In the generic JVM arguments, add the property-DubxInboundEndpointConfigPath=<Universal Behavior Exchange Subscriber Gateway endpoint install dir on the application server>. For example, add the property -DubxInboundEndpointConfigPath=C:\ubxInboundEndpoint.
4. Click **OK** to save the changes to the master configuration.
5. Restart the application server.

Deploy the endpoint in WebSphere Application Server.

1. Log in to the administrative console.

2. Navigate to **Applications** > **Application Types** > **Websphere enterprise applications**. Click **Install**.
3. Use the **Preparing for the application installation** option to locate the endpoint war file (`ubxInboundEndpoint.war`) to be installed and then click **Next**.
4. Click **Next** in subsequent pages to reach **Map context roots for Web modules**.
5. Use the **Map context roots for Web modules** to locate the Context Root and change value to `/UBXEndPoint`, this becomes the context root. Click **next**.
6. Click **Finish**.
7. Once the application finished installing, click **Save** to keep the changes on the master configuration.
8. Back in the listed and installed applications, mark the checkbox for ubxInboundEndpoint_war and click **Start** to load.

Configuring the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint with IBM MQ (optional)

By default, the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint works with ActiveMQ. Use the following instructions to configure the endpoint with IBM MQ.

Preparing the IBM MQ JAR files:

The client that runs the endpoint must have certain IBM MQ JAR files available in order for the connection factories to work.

If IBM MQ is already installed on the endpoint machine, the JAR files you need are already packaged with the IBM MQ installation. Add the following two JAR files to the system-level CLASSPATH environment variable. In Windows, the JAR files are automatically added to the classpath when IBM MQ is installed.

```
[MQ_HOME]\java\bin\com.ibm.mq.jar
[MQ_HOME]\java\bin\com.ibm.mqjms.jar
```

If IBM MQ is not installed on the machine, you should instead copy com.ibm.mq.allclient.jar and jms.jar from your MQ server to your endpoint server and manually add them to CLASSPATH.

For more information about installing or relocating IBM MQ JAR files, see http://www.ibm.com/support/docview.wss?uid=swg21376217.

Your application server needs to be running Java 1.7 or higher, as IBM MQ v8 JAR files do not support Java 1.6.

WebSphere Application Server comes pre-packaged with IBM MQ support and does not require any additional JAR files.

Configuring the endpoint

1. Go to the <endpoint install dir on the application server> directory.
2. Back up or rename `ubxInboundEndpoint-spring.xml` and `ubxInboundEndpoint.properties`.
3. Navigate to the `IBMMQ` subdirectory. It will contain alternate versions of the above files.
4. Add your MQ server connection information to this version of `ubxInboundEndpoint.properties`.

5. Copy ubxInboundEndpoint-spring.xml andubxInboundEndpoint.properties from /ubxInbdoundEndpoint/IBMMQ to the main/ubxInboundEndpoint directory.

Configuring the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint ubxInboundEndpoint.properties file

Use the ubxInboundEndpoint.properties file to configure where to send Universal Behavior Exchange event payload to IBM Interact Inbound Gateway for IBM Universal Behavior Exchange. The ubxInboundEndpoint.properties file is in the <gateway endpoint install dir on the application server> directory.

jmsBrokerUrl
Required - The JMS queue information where the producer writes the data.

jmsMaximumRetries
Required - The maximum number of retries to send a message to the JMS queue.

jmsRetryDelay
Required - The redelivery delay in milliseconds.

maximumEndPointThreadPoolSize
Required - The maximum number of threads for the thread pool to handle IBM Universal Behavior Exchange event data and write to JMS queue. This integer number defines the size of the thread pool.

clientIDFieldName
Optional - The field name used in the payload for the client id (sub category). A sub category is used when this program is running on multiple instances of the same product. For example: clientIDFieldName=clientID

A restart of the gateway endpoint webapp (ubxInboundEndpoint.war) is required in web server or application server for any changes in this file to take effect.

Configuring the IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint inboundProducerNameConfig.properties file (optional)

The IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint sends the event to Interact by writing to a JMS queue. The default event message uses the producer name value UBX. Use the inboundProducerNameConfig.properties file to override the producer name based on the UBX source field value from the payload. This is typically the UBX endpoint name. The inboundProducerNameConfig.properties file is in the <gateway endpoint install dir on the application server> directory.

SOURCE.{UBX source name}={producer name}
Example: SOURCE.CustomerAEndpoint=UBX-CustomerAEndpoint.

A restart of the gateway endpoint webapp (ubxInboundEndpoint.war) is required in web server or application server for any changes in this file to take effect.

Configuring the gateway endpoint inboundQueueNameConfig.properties file (optional)

The IBM Interact Inbound Gateway for IBM Universal Behavior Exchange endpoint sends the event to Interact by writing to a JMS queue. The default queue name is the same as the producer name. Use the inboundQueueNameConfig.properties file

to override the default JMS queue name by the producer name. The default producer name is UBX unless it's overridden in the inboundQueueNameConfig.properties file. The inboundProducerNameConfig.properties file is in the <gateway endpoint install dir on the application server> directory.

{producer name}={JMS queue name}

```
Example:
UBX=UBXInboundQueue.
UBX-CustomerAEndpoint=UBX-CustomerAEndpointQueue
```

A restart of gateway endpoint webapp (ubxInboundEndpoint.war) is required in web server or application server for any changes in this file to take effect.

Configuring the gateway endpoint log4j.properties file

Use the log4j.properties file to configure different log level for the endpoint. The log4j.properties file is in the <gateway endpoint install dir on the application server> directory.

Description

Set the log level for log4j.logger.com.ibm.x1solution.jms.producer, log4j.logger.com.ibm.web.offerorchestration.inbound.common and log4j.logger.com.ibm.web.offerorchestration.inbound.ubx accordingly.

Configuring the interactEventNameMapping.properties file

Use this file to map the value of the payload event field that is defined in the interactEventPayloadMapping.properties file as [EventName] to the Interact event name. The fallback is to use the event name as it comes in with the Universal Behavior Exchange event payload. The interactEventNameMapping.properties file is in the <Install dir>\conf\inbound\UBX directory.

{UBX event name}={Interact event name}

Example: matchedIdentity=recommendedOfferEven

If support for payload data from specific source is necessary, this file may also be placed in the <Install dir>\conf\inbound\UBX\{source} directory. The value for source should match the value of source field in the Universal Behavior Exchange event payload, typically the Universal Behavior Exchange endpoint name. If support for data using specific versions is necessary, this file may also be placed in the <Install dir>\conf\inbound\UBX\{source}\version-{version} directory. The value for version should match the value of version field in the Universal Behavior Exchange event payload. To support multiple Universal Behavior Exchange instance data, this file may also be placed in the <Install dir>\conf\inbound\UBX\{source}\version-{version}\account-{clientID} directory. The value for clientID should match the value of clientID in the Universal Behavior Exchange event payload.

Configuring the interactEventPayloadMapping.properties file

Use the interactEventPayloadMapping.properties file to map the inbound field to the Interact API parameters. The interactEventPayloadMapping.properties file is in the <Install dir>\conf\inbound\UBX directory.

Interact API parameters: The value must start with a field type definition, followed by either a static value when the value is in double quotes, or a field name from

the payload data. (FIELD_TYPE)"STATIC_VALUE" or (FIELD_TYPE)PAYLOAD_FIELD_NAME. FIELD_TYPE can be either `String`, `Numeric`, or `DateTime`.

```
Example:
[SessionID]=(String)interactprofileid
[EventName]=(String)code
[AudienceIDFieldNames]=(String)"change_me"
[AudienceIDFieldValues]=(String)interactprofileid
[AudienceLevel]=(String)"change_me"
[InteractChannel]=(String)"change_me"
```

Event data: These properties are used to map the event attributes that can be used in your outbound channel communications. The left side contains the variable names you use in your outbound channel communication.

The value must start with a field type definition, followed by either a static value when the value is in double quotes, or a field name from the payload data. (FIELD_TYPE)"STATIC_VALUE" or (FIELD_TYPE)PAYLOAD_FIELD_NAME. FIELD_TYPE can be either `String`, `Numeric`, or `DateTime`.

If support for payload data from specific source is necessary, this file may also be placed in the <Install dir>\conf\inbound\UBX\{source} directory. The value for source should match the value of source field in the Universal Behavior Exchange event payload, typically the Universal Behavior Exchange endpoint name. If support for data using specific versions is necessary, this file may also be placed in the <Install dir>\conf\inbound\UBX\{source}\version-{version} directory. The value for version should match the value of version field in the Universal Behavior Exchange event payload. To support multiple Universal Behavior Exchange instance data, this file may also be placed in the <Install dir>\conf\inbound\UBX\{source}\version-{version}\account-{clientID} directory. The value for clientID should match the value of clientID in the Universal Behavior Exchange event payload.

Creating an endpoint and event in UBX

This is a sample endpoint and event that you can use as an example.

Use the following steps to create an endpoint and event in UBX.

1. Use the REST API client to post the requests to UBX.
2. Register an endpoint in UBX with JSON. See the following example.

```
Method Call: PUT
URL: https://ubx-qa1-api.adm01.com/v1/endpoint
Headers:
Content-Type: application/json
Accept-Charset: UTF-8
Authorization: Bearer 912586bf-190d-48f9-8488-26f1bf532ef3
(Note: This is the Auth Key generated from the UBX UI.)
Body
 {
   "name":"Interactubxdk1",
   "description":"Interactubxdk1",
   "providerName":"IBM", "
   "url":"http://169.38.71.122:9081/ubxEndPoint/UBXEndPoint",
   "endpointTypes":{
     "event":{
       "source":{
         "enabled":true
       },
       "destination":{
```

```
        "enabled":true,
        "url":"http://169.38.71.122:9081/UBXEndPoint/UBXEndPoint",
        "destinationType":"push"
       }
     }
},
"marketingDatabasesDefinition":{
  "marketingDatabases":[
    {
      "name":"IDSync",
      "identifiers":[
        {
      "name":"interactprofileid",
      "type":"INTERACTID"
          }
        ]
     }
  ]
}
}
```

3. Register an eventtype in UBX with JSON. See the following example.

```
Event Registration for Interact Event in UBX
Method Call: POST
URL: https://ubx-qa1-api.adm01.com/v1/eventtype

Headers:
Content-Type: application/json
Accept-Charset: UTF-8
Authorization: Bearer 912586bf-190d-48f9-8488-26f1bf532ef3
Note: This is the Auth Key generated from the UBX UI.)
Bearer 912586bf-190d-48f9-8488-26f1bf532ef3
Body
{
   "name": "recommendedOffers",
   "description": "recommended offers by OMO",
   "code": "recommendedOffers"
}
```

4. Post an event to UBX with JSON. See the following example.

```
{
  "channel" : "mobile",
  "identifiers" : [
    {
      "name" : "interactprofileid",
      "value" : "55"
    }
  ],
"events" : [
  {
    "code" : "recommendedOffers",
    "timestamp" : "2015-12-28T20:16:12Z"
   }
  ]
}
```

Using the IBM Interact Outbound Gateway for IBM Universal Behavior Exchange

To use the IBM Interact Outbound Gateway for IBM Universal Behavior Exchange, you must configure Interact, UBX, and the gateway.

Use the following configurations as an example for your configuration.

If you use UBX as an outbound channel, Interact acts as publisher type of endpoint, which publish events to UBX. From UBX these events can be sent to subscriber.

Before you begin the configuration, request for outbound access to host machine. You need net access to be enabled for the host machine.

You can download the gateway from http://www.ibm.com/support/fixcentral/swg/quickorder?parent=Enterprise%2BMarketing%2BManagement&product=ibm/Other+software/Unica+Interact&release=All&platform=All&function=fixId&fixids=IBM_Interact_OMO_Gateway_for_UBX_Publisher_2.0&includeRequisites=1&includeSupersedes=0&downloadMethod=http&source=fc.

Registering endpoints and events in UBX

1. From UBX, navigate to the **EndPoints** tab. Click **Register new endpoint** to get an auth key. The auth key generated from UBX should be used for publisher endpoint and adding event. For the subscriber endpoint, the new auth key should be generated from UBX. Make note of the key.
2. Register your publisher endpoint.
   a. Open the REST API client tool.
   b. Select the method as `PUT`.
   c. Pass the headers as

      ```
      Content-Type : application/json
      Accept-Charset : UTF-8
      Authorization : Bearer 520301d7-7855-4ea7-b19d-0b395c1e6ae4
      (authKey generated in UBX)
      ```

   d. Pass the URL as

      ```
      URL: https://ubx-qa1-api.adm01.com/v1/endpoint
      ```

   e. For the body, pass the appropriate name for the publisher endpoint.

      For example:

      ```
      {
          "name":"Interact_Publisher",
          "description":"Endpoint for server created on 30thJan",
          "providerName":"IBM",    "url":"",
          "endpointTypes":{
              "event":{
              "source":{
                 "enabled":true
              }
          }
      },
      "marketingDatabasesDefinition":{
          "marketingDatabases":[
            {
                "name":"IDSync",
                "identifiers":[
                  {
                "name":"interactprofileid",
           "type":"INTERACTID"
              }
            ]
          }
        ]
      }
      }
      ```

3. Register your event. Make note of the [Event] code passed in body. This needs to be mapped in the `ubxContentMapping.properties` file. This is case sensitive.

a. Open the REST API client tool.
b. Select the method as `POST`.
c. Pass the same headers you used for your endpoint in the previous step.
d. Pass the URL as

```
URL: https://ubx-qa1-api.adm01.com/v1/eventtype
```

e. For the body, pass the appropriate name for the event.

For example:

```
{
   "name": "recommendedOffer",
   "description": "recommended
      contact frm UBX",    "code":
      "recommendedOffer"}
```

Note: The Event code that is passed must be mapped in the ubxContentMapping.properties file. The event code is case sensitive.

4. Add the subscriber endpoint.
   a. Open the REST API client tool.
   b. Select the method as `PUT`.
   c. Pass the same headers you used for your endpoint in the previous step.
   d. For registering the subscriber endpoint, create a new auth key in UBX.
   e. Pass the URL as

```
URL: https://ubx-qa1-api.adm01.com/v1/endpoint
```

   f. For the body, pass the appropriate name for the publisher endpoint.

For example:

```
{
   "name":"UBX_Subscriber",
   "description":"UBX_Subscriber for Subscribing Events ",
   "providerName":"IBM",
   "url":"http://ubxeventconsumer.mybluemix.net/ubxeventconsumer",
   "endpointTypes":{
      "event":{
        "source":{
          "enabled":true
        },
      "destination":{
        "enabled":true,
         "url":"http://ubxeventconsumer.mybluemix.net
          /ubxeventconsumer",
      }
   }
},
"marketingDatabasesDefinition":   {
  "marketingDatabases":[
    {
       "name":"IDSync",
       "identifiers":[
          {
       "name":"interactprofileid",
     "type":"INTERACTID"
          }
       ]
   }
 ]
}
```

5. After adding publisher and subscriber endpoints and event, you must subscribe the events from the publisher to the subscriber in UBX .
   a. In UBX, click **Subscribe to events** on the **Events** tab.

b. Select the event and destination.
c. Click **Subscribe**.

Configuring Interact and the gateway

1. Add the UBX gateway under the **Interact | triggeredMessage | gateways** configuration property. Set **ClassPath** to `file:///root/opt/OMO/lib/OMO_OutboundGateway_UBX.jar` and **ClassName** to

   ```
   com.ibm.interact.offerorchestration.outboundgateway.ubx.
   UBXOutboundGateway
   ```

2. Unzip the `OMO_OutboundGateway_UBX.zip` file on your host machine and point to the UBX jar from the extracted path.
3. Add `OMO-conf_outbound_common_httpConnectionConfig` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to `file:///opt/Interact<version>/Interact/OMO/conf/outbound/common/httpConnectionConfig.properties`. This is the Interact installation directory. The gateway installer downloads the gateway directory to the Interact installed directory.

   In the `httpConnectionConfig.properties` file in the Interact folder, specify the timeout.

   For example:

   ```
   connectTimeoutMs=180000
   ```

4. Add `OMO-conf_outbound_ubx_ubxConfig` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to the path of the `ubxConfig.properties` file in the Interact folder.

   In the `ubxConfig.properties` file, specify the `ubxURL`, `authKey`, and `interactProfileIdFieldName`.

   For example:

   ```
   authKey=912586bf-190d-48f9-8488-26f1bf532ef3
   [Auth Key used to register publisher endpoint and event in UBX]
   interactProfileIdFieldName=interactprofileid
   [Field name from the ubxContentMapping.properties file]
   ```

5. Add `OMO-conf_outbound_ubx_ubxContentAdditionalAttributes` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to the path of the `ubxContentAdditionalAttributes.properties` file in the Interact folder.
6. Add `OMO-conf_outbound_ubx_ubxContentMapping` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to the path of the `ubxContentMapping.properties` file in the Interact folder.

   Update the alues for `interactprofileid` and `eventName` in the `ubxContentMapping.properties` file.

   You can Pass Event Name in 3 formats: when the value is in double quotes, it is a static value; when the value is in the `offer.offerAttributeName` format, it maps to the offer attribute offerAttributeName; and when the value is in the `profile.profileAttributeName` format, it maps to the profile attribute profileAttributeName. The Event Name value should match the code used to register the event in UBX . This is case sensitive.

   For example:

   ```
   eventName="abandoned_shopping_carts"
   eventName=offer.Card
   eventName=profile.EMAIL
   ```

7. Add a channel under the **Interact | triggeredMessage | channel** configuration property.

8. Define the same channel in design time under **Campaign | partitions | partition [n] |Interact | outboundChannels**
9. Restart the application server.
10. Create a triggered messages rule with an event name and that uses the channel you added in the previous steps.
11. Deploy the interactive channel.
12. From the API Test client, start the session for interactive channel where triggered message rule is configured and the post event which triggers the offer to UBX.

Using IBM Interact Outbound Gateway for IBM Mobile Push Notification

To use this mobile push outbound or publisher gateway, you must configure Interact, IBM Marketing Cloud, and the gateway.

Use the following configurations as an example for your configuration.

You can download this gateway from https://www-945.ibm.com/support/fixcentral/swg/downloadFixes

Configuring IBM Marketing Cloud

1. Make sure you have an IBM Marketing Cloud account with push access. Also make note of your Client ID, Client Secret, and Refresh Token.
2. On the **Data** tab, create a new database. Add a new `Mobile User ID` to the database along with the default fields.
3. On the **Search** tab, search by the `Mobile User ID` field. Hover the mouse key on first `No email` field. You will see the `recipient ID` at the bottom of browser window. Add this `recipient ID` to the Interact profile table.

Configuring the IBM Interact Outbound Gateway for IBM Mobile Push Notification

1. Download and install the mobile push outbound gateway from https://www-945.ibm.com/support/fixcentral/swg/downloadFixes
2. Configure the `silverpopEngagePushConfig.properties` file.

   For example:

   ```
   OauthServiceURL=https://apipilot.silverpop.com/oauth/token
   pushServiceURL=https://apipilot.silverpop.com/rest/channels/push/sends
   ```

3. Configure the `silverpopEngagePushContentMapping.properties` file.

   For example:

   ```
   Interact Profile table attributes:
   appKey=appKey
   engageRecipientId=recipientId
   mobileUserId=mobileUserId
   deviceType=deviceType

   Interact Offer attributes:
   simpleSubject=simpleSubjectAttr
   simpleMessage=simpleMessageAttr
   simpleActionData=simpleActionDataAttr
   simpleActionType=simpleActionTypeAttr
   simpleActionLabel=simpleActionLabelAttr
   personalizeAttributeList=personalizeAttributeList
   contentId=ContentID
   campaignId=campaignId
   ```

Configuring Interact

1. Create the following offer attributes.

   ```
   simpleActionDataAttr: string
   simpleActionLabelAttr: String
   simpleActionTypeAttr: string
   simpleMessageAttr: string
   simpleSubjectAttr: string
   contentID: string
   campaignId=string
   personalizeAttributeList=string
   ```

2. Create an offer template with the offer attributes and the following offer values.

   ```
   simpleActionDataAttr: www.ibm.com
   simpleActionLabelAttr: Open URL
   simpleActionTypeAttr: url
   simpleMessageAttr: <Enter your message text here>
   simpleSubjectAttr: <Enter subject here>
   contentID: ID of the push message template that is created in Engage.
   PersonalizeAttributeList: A comma separated list of attribute name
   value pairs that you want to put in the personalizationDefaults
   section of the payload to be sent to Engage.
   ```

   When you use the `contentID` attribute, the other `simple..` attributes are ignored as the complete details are picked up from the Engage template.

   Example personalizedAttributeList

   ```
   personalizeAttributeList=discount=10,Offercost=20
   campaignId=campaignname that you want to use for this campaign.
   ```

3. Your profile table has the following columns and values.

   ```
   appKey: gcsTQo6v79
   recipientId: 13472242
   deviceType: android or ios
   ```

4. Add the gateway under the **Interact | triggeredMessage | gateways** configuration property. Set **ClassName** to

   ```
   com.ibm.interact.offerorchestration.outboundgateway.silverpop.engage.push.
   SilverpopEngagePushOutboundGateway
   ```

   Set the **ClassPath** to `file://<EngagePushGateway_home_dir>/lib/OMO_OutboundGateway_Silverpop_Engage_Push.jar`.

5. Add `OMO-silverpopEngagePushConfig` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to the file path of your `silverpopEngagePushConfig.properties` file.

6. Add `OMO-silverpopEngagePushContentMapping` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to the file path of your `silverpopEngagePushContentMapping.properties` file.

7. Add `OMO-conf_outbound_common_httpConnectionConfig` as a parameter under **Interact | triggeredMessage | gateways | [gatewayName] | Parameter Data**. Set the **value** to the file path of your `httpConnectionConfig.properties` file.

   In the `httpConnectionConfig.properties` file in the Interact folder, specify the timeout.

   For example:

   ```
   connectTimeoutMs=6000
   ```

8. Create a channel and a handler under **Interact | triggeredMessage** and use the [Mobile_Push] gateway that you created above in that channel. This channel is used in the triggred message to send push messages.

9. Create an interactive channel and add a triggered message that uses the offer you created previously to the trigger rule.
10. Deploy the interactive channel.
11. From the API Test client, perform a `startSession` for interactive channel where triggered message rule is configured and the `postEvent` which triggers the offer to Mobile Push.
12. Check the Interact logs to make sure the push was sent successfully. The status code `202` means successful delivery.

Using the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud

You can use this integration with Silverpop, Interact and IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud to send triggered email offers to your customers.

Be sure that the following prerequisites are true.

- Create a customer audience profile table with an email column. Use this profile table for your interactive channel.
- Request that net access to the host machine is enabled for your outbound channel.
- Copy and extract the `OMO_OutboundGateway_Silverpop.zip` file on host machine

You can download the gateway from http://www.ibm.com/support/fixcentral/swg/quickorder?parent=Enterprise%2BMarketing%2BManagement&product=ibm/Other+software/Unica+Interact&release=All&platform=All&function=fixId&fixids=IBM_Interact_OMO_OutboundGateway_Silverpop_2.0&includeRequisites=1&includeSupersedes=0&downloadMethod=http&source=fc.

Adding a dispatcher for the gateway integration

The dispatcher adds your offer into a queue for the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud so that your offer email can be sent.

You must add a dispatcher to use the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud.

1. Navigate to **Interact | triggeredMessage | dispatchers | <dispatcherName>** in configuration properties.
2. Add a **New category name** for your dispatcher.
3. Select a **type**. You can choose from InMemoryQueue, JMSQueue, and Custom.
4. Enter the **className**.
5. Enter the **classPath**.

Adding a gateway for the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud

In the integration, the gateway sends eligible offers to your customers by email.

You must add a gateway for the integration.

Note: Interact does not support multiple instances of the same gateway.

1. Navigate to **Interact | triggeredMessage | gateways | <gatewayName>** in configuration properties.

2. Add a **New category name** for your gateway.
3. Set the **className** to the following path.

   ```
   com.ibm.interact.offerorchestration.outboundgateway.
   silverpop.SilverpopEmailOutboundGateway
   ```

4. Set the **classPath** to the location of the outbound gateway jar path from the extracted folder.

   For example:

   ```
   file:///opt/OMO_SilverPop/OMO_OutboundGateway_Silverpop/lib/
   OMO_OutboundGateway_Silverpop.jar
   ```

5. Add the following parameters to your gateway.

   ```
   OMO-conf_outbound_common_httpConnectionConfig
   OMO-conf_outbound_silverpop_silverpopConfig
   OMO-conf_outbound_silverpop_silverpopContentMapping
   deliveryTimeoutMillis
   ```

Configuring the OMO-conf_outbound_common_httpConnectionConfig parameter

You must configure the OMO-conf_outbound_common_httpConnectionConfig parameter for your gateway.

1. Navigate to **Interact | triggeredMessage | gateways | <SilverpopGatewayName> | OMO-conf_outbound_common_httpConnectionConfig** in configuration properties.
2. Set the **value** to `file:///opt/Interact<version>/Interact/OMO/conf/outbound/common/httpConnectionConfig.properties`. This is the Interact installation directory. The Interact installer downloads the `httpConnectionConfig.properties` file to the Interact installation directory.
3. In the `httpConnectionConfig.properties` file in the Interact folder, specify the timeout.

   For example:

   ```
   connectTimeoutMs=60000
   ```

Configuring the OMO-conf_outbound_silverpop_silverpopConfig parameter

You must configure the OMO-conf_outbound_silverpop_silverpopConfig parameter for your gateway.

1. Navigate to **Interact | triggeredMessage | gateways | <SilverpopGatewayName> | OMO-conf_outbound_silverpop_silverpopConfig** in configuration properties.
2. Set the **value** to the path of the `silverpopConfig.properties` file in your `OMO_OutboundGateway_silverpop` folder.

   For example:

   ```
   file:///opt/OMO_SilverPop/OMO_OutboundGateway_Silverpop/conf/outbound/
   silverpop/silverpopConfig.properties
   ```

3. In the `silverpopConfig.properties` file in the extracted `OMO_OutboundGateway_Silverpop.zip` folder, set values for `OauthServiceURL`, `xmlAPIServiceURL`, `clientID`, `clientSecret`, and `refreshToken`. Consult your Marketing Cloud administrator to get customer specific values from the `transact.xml` file.

Configuring the OMO-conf_outbound_silverpop_silverpop ContentMapping parameter

You must configure the OMO-conf_outbound_silverpop_silverpopContentMapping parameter for your gateway.

1. Navigate to **Interact | triggeredMessage | gateways | <SilverpopGatewayName> | OMO-conf_outbound_silverpop_silverpopContentMapping** in configuration properties.
2. Set the **value** to the path of the `silverpopContentMapping.properties` file in your `OMO_OutboundGateway_silverpop` folder.
3. In the `silverpopContentMapping.properties` file in the `OMO_OutboundGateway_Silverpop.zip` folder, set the values for your content mapping.
   a. Set the `campaignId` property. The value for this property is an offer attribute name that is specified in your offer templates.
   b. Set the `email` property. The value for this property is the column name in your profile table. Add an `email`column in your profile table and specify the email IDs. These are the mail IDs of the recipients.
   c. Define your offer attributes in `additionalOfferPfAttributesUsedInEmail`. This property sets the attributes from your offer template that are needed for the mailing template. You can use `additionalProfilePfAttributesUsedInEmail` to define fields from your profile table. You can use `*` to consider all offer attributes and column values.

Configuring the deliveryTimeoutMillis parameter

To increase the Interact server timeout to connect with Marketing Cloud server, set the `deliveryTimeoutMills` parameter.

1. Navigate to **Interact | triggeredMessage | gateways | <SilverpopGatewayName> | deliveryTimeoutMillis** in configuration properties.
2. Set the **value**. For example, you could set **value** to 60000. This would increase the server timeout to 60000 milliseconds.

Add a channel handler for the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud

Add a channel handler in the Interact runtime environment.

1. Navigate to **Interact | triggeredMessage | channels | <SilverpopChannelName> | <handlerName>** in configuration properties.
2. Add a **New category name** for your channel handler.
3. Set the name of the dispatcher you previously added.
4. Set the name of the gateway you previously added.
5. Set the **mode**. If Failover is selected, this handler is used only when all the handlers with higher priorities defined within this channel failed to send offers. If **Addon** is selected, this handler is used no matter if other handlers have successfully sent offers.
6. Set the **priority** for this handler.

Adding an outbound channel for the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud

Add an outbound channel in the Interact design environment.

1. Navigate to **Campaign | partitions | partition[n] | Interact | outboundChannels** in configuration properties.
2. Add a **New category name** for your outbound channel.

3. Add a **name** for your outbound channel. Make sure the channel name is the same as the channel name you added in the **Interact | triggeredMessage | channels | <SilverpopChannelName>** configuration property.

Configuring the transactional mailing with the IBM Interact Email (Transact) Outbound Gateway for IBM Marketing Cloud

You must configure your transactional mailing to send your email offer.

1. In the Marketing Cloud (Transact), click **Data** > **Create Database**. Then click **Create** to create a profile table. You can also import the profile table where you added the email column.
2. Click **Automation** > **Transactional messages** > **Create Group**. Select **Transact** for the **Event Trigger**. You also need to select the datasource you previously created. Click **Save & Activate**.

   The offer that is sent through The Marketing Cloud should have the same attribute you set for the `campaignId` in the `silverpopContentMapping.properties` file. The value for this offer attribute is the `campaignId` that is generated for the automated message group.
3. Click **Content** > **Create Mailings** and select the content source from the previous step. Enter the mailing body. Click **Automate**. Select **Assign Mailing to Existing Group of Automated Messages**. Click **Submit & Activate**.

   The mailing subject line and body can be personalized using offer attributes and profile attributes. Use the `%%Attribute_Name%%` syntax to define attributes.
4. The Marketing Cloud server only accepts outbound gateways submissions from IP addresses set up in advance. To add an IP address, navigate to **Settings** > **Org Admin** > **Security Settings** > **Access Restrictions**.
5. If you use the WebSphere Application Server, you need to import the Marketing Cloud SSL certificate. This is not required for WebLogic users.
   a. In the WebSphere Application Server console, navigate to **SSL certificate and key managemen** > **Key stores and certificate** > **NodeDefaultTrustStore** > **Signer certificates** > **Retrieve from port**.
   b. Set the host and port.
   c. Restart the WebSphere Application Server.

Before you contact IBM technical support

If you encounter a problem that you cannot resolve by consulting the documentation, your company's designated support contact can log a call with IBM technical support. Use these guidelines to ensure that your problem is resolved efficiently and successfully.

If you are not a designated support contact at your company, contact your IBM administrator for information.

Note: Technical Support does not write or create API scripts. For assistance in implementing our API offerings, contact IBM Professional Services.

Information to gather

Before you contact IBM technical support, gather the following information:

- A brief description of the nature of your issue.
- Detailed error messages that you see when the issue occurs.
- Detailed steps to reproduce the issue.
- Related log files, session files, configuration files, and data files.
- Information about your product and system environment, which you can obtain as described in "System information."

System information

When you call IBM technical support, you might be asked to provide information about your environment.

If your problem does not prevent you from logging in, much of this information is available on the About page, which provides information about your installed IBM applications.

You can access the About page by selecting **Help > About**. If the About page is not accessible, check for a `version.txt` file that is located under the installation directory for your application.

Contact information for IBM technical support

For ways to contact IBM technical support, see the IBM Product Technical Support website: (http://www.ibm.com/support/entry/portal/open_service_request).

Note: To enter a support request, you must log in with an IBM account. This account must be linked to your IBM customer number. To learn more about associating your account with your IBM customer number, see **Support Resources > Entitled Software Support** on the Support Portal.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
B1WA LKG1
550 King Street
Littleton, MA 01460-1250
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Privacy Policy and Terms of Use Considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. A cookie is a piece of data that a web site can send to your browser, which may then be stored on your computer as a tag that identifies your computer. In many cases, no personal information is collected by these cookies. If a Software Offering you are using enables you to collect personal information through cookies and similar technologies, we inform you about the specifics below.

Depending upon the configurations deployed, this Software Offering may use session and persistent cookies that collect each user's user name, and other personal information for purposes of session management, enhanced user usability, or other usage tracking or functional purposes. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

Various jurisdictions regulate the collection of personal information through cookies and similar technologies. If the configurations deployed for this Software Offering provide you as customer the ability to collect personal information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for providing notice and consent where appropriate.

IBM requires that Clients (1) provide a clear and conspicuous link to Customer's website terms of use (e.g. privacy policy) which includes a link to IBM's and Client's data collection and use practices, (2) notify that cookies and clear gifs/web beacons are being placed on the visitor's computer by IBM on the Client's behalf along with an explanation of the purpose of such technology, and (3) to the extent required by law, obtain consent from website visitors prior to the placement of cookies and clear gifs/web beacons placed by Client or IBM on Client's behalf on website visitor's devices

For more information about the use of various technologies, including cookies, for these purposes, See IBM's Online Privacy Statement at: http://www.ibm.com/privacy/details/us/en section entitled "Cookies, Web Beacons and Other Technologies."

Printed in USA