IBM Unica Interact
Version 8 Release 6
May 25, 2012

Administrator's Guide

Note
Before using this information and the product it supports, read the information in "Notices" on page 247.

This edition applies to version 8, release 5, modification 0 of IBM Unica Interact (product number 5725-D22) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Administering IBM Unica Interact 1

Interact key concepts 1
- Audience levels 1
- Design environment 2
- Events 2
- Interactive channels 2
- Interactive flowcharts 2
- Interaction points 3
- Offers. 3
- Profiles 3
- Runtime environment 3
- Runtime sessions 4
- Touchpoints 4
- Treatment rules 4

Interact architecture 5
Interact network considerations 5

Chapter 2. Configuring IBM Unica Interact Users 7

Configuring the runtime environment user 7
Configuring design environment users. 7
Example design environment permissions 9

Chapter 3. Managing Interact data sources 11

Working with Interact data sources 11
Databases and the applications 12
Campaign system tables 12
Runtime tables 13
Test run tables 14
- Overriding the default data types used for dynamically created tables 15
- To override the default data types 15
- Default data types for dynamically created tables 16

Profile database 16
Learning tables 18
Contact history for cross-session response tracking 19
Using Interact feature scripts 19
About contact and response history tracking . . . 19
- Configuring contact and response types 20
- Additional response types 20
- Runtime environment staging tables to Campaign history tables mapping 22
- To configure JMX monitoring for the contact and response history module 24

About cross-session response tracking 24
- Cross-session response tracking data source configuration 25
- Configuring contact and response history tables for cross-session response tracking. 25
- To enable cross-session response tracking . . . 27
- Cross-session response offer matching 28

Using a database load utility with the runtime environment 30
- To enable a database load utility with runtime environment 31

Chapter 4. Offer serving 33

Offer eligibility 33
- Generating a list of candidate offers 33
- Calculating the marketing score 34
- Influencing learning 35

About suppressing offers 35
- To enable the offer suppression table 36
- Offer suppression table 36

Global offers and individual assignments 36
- To define the default cell codes 37
- To define the UACI_ICBatchOffers table 37
- About the global offers table. 37
- To enable the global offer table 38
- Global offer table 38
- About the score override table 40
- To enable score override table 40
- Score override table 40

Interact built-in learning overview 43
- Understanding Interact learning 43
- To enable the learning module 45
- Learning attributes 45
- To define a learning attribute 46
- To define dynamic learning attributes 47
- To enable external learning 48

Chapter 5. Understanding the Interact API 49

Interact API dataflow 49
Simple interaction planning example 52
Designing the Interact API integration 55
- Points to consider 55

Chapter 6. Managing the IBM Unica Interact API. 57

Locale and the Interact API 57
About JMX monitoring 57
- To configure Interact to use JMX monitoring with the RMI protocol 57
- To configure Interact to use JMX monitoring with the JMXMP protocol 58
- Using the jconsole scripts. 58
- JMX attributes 59
- JMX operations 66

Chapter 7. Classes and methods for the IBM Unica Interact API 67

Interact API Classes 67
- Java serialization over HTTP prerequisites . . . 67
- SOAP prerequisites 68

API JavaDoc . . . 68
About API examples . . . 68
Working with session data . . . 68
About the InteractAPI class . . . 69
endSession . . . 69
executeBatch . . . 70
getInstance . . . 71
getOffers . . . 72
getOffersForMultipleInteractionPoints . . . 73
getProfile . . . 75
getVersion . . . 76
postEvent . . . 77
setAudience . . . 79
setDebug . . . 80
startSession . . . 81
Reserved parameters . . . 84
About the AdvisoryMessage class . . . 85
getDetailMessage . . . 86
getMessage . . . 86
getMessageCode . . . 87
getStatusLevel . . . 87
About the AdvisoryMessageCode class . . . 87
Advisory message codes . . . 88
About the BatchResponse class . . . 89
getBatchStatusCode . . . 89
getResponses . . . 90
About the Command interface . . . 90
setAudienceID . . . 91
setAudienceLevel . . . 92
setDebug . . . 92
setEvent . . . 93
setEventParameters . . . 94
setGetOfferRequests . . . 95
setInteractiveChannel . . . 96
setInteractionPoint . . . 96
setMethodIdentifier . . . 96
setNumberRequested . . . 97
setRelyOnExistingSession . . . 98
About the NameValuePair interface . . . 98
getName . . . 98
getValueAsDate . . . 98
getValueAsNumeric . . . 99
getValueAsString . . . 99
getValueDataType . . . 100
setName . . . 100
setValueAsDate . . . 101
setValueAsNumeric . . . 101
setValueAsString . . . 101
setValueDataType . . . 102
About the Offer class . . . 102
getAdditionalAttributes . . . 103
getDescription . . . 103
getOfferCode . . . 104
getOfferName . . . 104
getScore . . . 104
getTreatmentCode . . . 105
About the OfferList class . . . 105
getDefaultString . . . 106
getRecommendedOffers . . . 106
About the Response class . . . 107
getAdvisoryMessages . . . 107
getApiVersion . . . 107
getOfferList . . . 108
getAllOfferLists . . . 108
getProfileRecord . . . 109
getSessionID . . . 109
getStatusCode . . . 109

Chapter 8. About the ExternalCallout API . . . 111
IAffiniumExternalCallout interface . . . 111
To add a web service for use with EXTERNALCALLOUT . . . 112
getNumberOfArguments . . . 112
getValue . . . 112
initialize . . . 113
shutdown . . . 113
ExternalCallout API example . . . 114
IInteractProfileDataService interface . . . 115
To add a data source for use with Profile Data Services . . . 115

Chapter 9. IBM Unica Interact Utilities 117
Run Deployment Utility (runDeployment.sh/.bat) 117

Chapter 10. About the Learning API 121
To enable external learning . . . 122
ILearning interface . . . 122
initialize . . . 123
logEvent . . . 123
optimizeRecommendList . . . 124
reinitialize . . . 124
shutdown . . . 125
IAudienceID interface . . . 126
getAudienceLevel . . . 126
getComponentNames . . . 126
getComponentValue . . . 126
IClientArgs . . . 126
getValue . . . 126
IInteractSession . . . 127
getAudienceId . . . 127
getSessionData . . . 127
IInteractSessionData interface . . . 127
getDataType . . . 127
getParameterNames . . . 127
getValue . . . 128
setValue . . . 128
ILearningAttribute . . . 128
getName . . . 128
ILearningConfig . . . 129
ILearningContext . . . 129
getLearningContext . . . 129
getResponseCode . . . 129
IOffer . . . 130
getCreateDate . . . 130
getEffectiveDateFlag . . . 130
getExpirationDateFlag . . . 130
getOfferAttributes . . . 130
getOfferCode . . . 131
getOfferDescription . . . 131
getOfferID . . . 131

getOfferName 131
getUpdateDate 131
IOfferAttributes 132
getParameterNames 132
getValue 132
IOfferCode interface 132
getPartCount 132
getParts 132
LearningException. 133
IScoreOverride 133
getOfferCode 133
getParameterNames 133
getValue 134
ISelectionMethod 134
ITreatment interface 134
getCellCode 134
getCellId 134
getCellName 135
getLearningScore 135
getMarketerScore 135
getOffer 135
getOverrideValues. 136
getPredicate 136
getPredicateScore 136
getScore 136
getTreatmentCode 137
setActualValueUsed 137
Learning API example 137

Appendix A. IBM Unica Interact WSDL 141

Appendix B. Interact runtime environment configuration properties . 149

Interact | general 149
Interact | general | learningTablesDataSource 149
Interact | general | prodUserDataSource . . . 151
Interact | general | systemTablesDataSource 152
Interact | general | testRunDataSource. . . . 157
Interact | general | contactAndResponseHistoryDataSource . . . 159
Interact | general | idsByType 160
Interact | flowchart 160
Interact | flowchart | ExternalCallouts | [ExternalCalloutName] 162
Interact | flowchart | ExternalCallouts | [ExternalCalloutName] | Parameter Data | [parameterName] 163
Interact | monitoring. 163
Interact | profile 164
Interact | profile | Audience Levels | [AudienceLevelName] 165
Interact | profile | Audience Levels | [AudienceLevelName] | Offers by Raw SQL . . 168
Interact | profile | Audience Levels | [AudienceLevelName | Profile Data Services | [DataSource]. 169
Interact | offerserving 171
Interact | offerserving | Built-in Learning Config. 171
Interact | offerserving | External Learning Config. 172
Interact | offerserving | External Learning Config | Parameter Data | [parameterName] . 173
Interact | services 173
Interact | services | contactHist 174
Interact | services | contactHist | cache . . . 174
Interact | services | contactHist | fileCache . . 175
Interact | services | defaultedStats 175
Interact | services | defaultedStats | cache . . 175
Interact | services | eligOpsStats. 176
Interact | services | eligOpsStats | cache . . . 176
Interact | services | eventActivity 177
Interact | services | eventActivity | cache . . 177
Interact | services | customLogger 177
Interact | services | customLogger | cache . . 178
Interact | services | responseHist 178
Interact | services | responseHist | cache. . . 179
Interact | services | responseHist | fileCache 179
Interact | services | crossSessionResponse . . 180
Interact | services | crossSessionResponse | cache 180
Interact | services | crossSessionResponse | OverridePerAudience | [AudienceLevel] | TrackingCodes | byTreatmentCode 181
Interact | services | crossSessionResponse | OverridePerAudience | [AudienceLevel] | TrackingCodes | byOfferCode. 182
Interact | services | crossSessionResponse | OverridePerAudience | [AudienceLevel] | TrackingCodes | byAlternateCode 183
Interact | services | threadManagement | contactAndResponseHist 184
Interact | services | threadManagement | allOtherServices 185
Interact | services | threadManagement | flushCacheToDB 186
Interact | sessionManagement. 187

Appendix C. Interact design environment configuration properties . 189

Campaign | partitions | partition[n] | reports . . 189
Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking. 191
Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking | runtimeDataSources | [runtimeDataSource] . . 195
Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking | contactTypeMappings 196
Campaign | partitions | partition[n] | Interact | contactAndResponseHistTracking | responseTypeMappings 196
Campaign | partitions | partition[n] | Interact | report 197
Campaign | partitions | partition[n] | Interact | learning 197
Campaign | partitions | partition[n] | Interact | learning | learningAttributes | [learningAttribute]. 200

Campaign | partitions | partition[n] | Interact | deployment 201
Campaign | partitions | partition[n] | Interact | serverGroups | [serverGroup] 201
Campaign | partitions | partition[n] | Interact | serverGroups | [serverGroup] | instanceURLs | [instanceURL] 201
Campaign | partitions | partition[n] | Interact | flowchart 202
Campaign | partitions | partition[n] | Interact | whiteList | [AudienceLevel] | DefaultOffers . . . 202
Campaign | partitions | partition[n] | Interact | whiteList | [AudienceLevel] | offersBySQL . . . 203
Campaign | partitions | partition[n] | Interact | whiteList | [AudienceLevel] | ScoreOverride . . 203
Campaign | partitions | partition[n] | server | internal 204
Campaign | monitoring 206

Appendix D. Real-time offer personalization on the client side. . . 209
About the Interact Message Connector 209
Installing the Message Connector 210
Creating the Message Connector links 216
About the Interact Web Connector 218
Installing the Web Connector on the runtime server 219
Installing the Web Connector as a separate web application 220
Configuring the Web Connector 221
Using the Web Connector Admin Page 232
Sample Web Connector Page 233

Appendix E. Interact and Intelligent Offer Integrated Product Recommendations 237
Overview of Interact integration with Intelligent Offer 237
Integration Prerequisites 238
Configuring an offer for Intelligent Offer integration 238
Using the Integration Sample Project 239

Contacting IBM Unica technical support 245

Notices 247
Trademarks 249

Chapter 1. Administering IBM Unica Interact

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 regarding how Interact works you should understand to make your tasks easier. The sections which follow describe the administrative tasks associated with Interact.

The second part of this guide describes the Application Programming Interfaces (API) available with Interact: Interact API, ExternalCallout API, and the Learning API.

Interact key concepts

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. For example, a set of campaigns could 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 as well as 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 may need to exist for businesses, companies, divisions, groups, individuals, accounts, and so on.

These audience levels may 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

The design environment is where you perform the majority of your Interact configuration. In the design environment, you define events, interaction points, smart segments, and treatment rules. After configuring 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, taken by a visitor, which triggers an action in the runtime environment, such as 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 using the `postEvent` method. An event can lead to one or more of the following actions defined in the Interact design environment:

- Trigger re-segmentation
- Log offer contact
- Log offer acceptance
- Log offer rejection

You can also use events to trigger actions 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

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 involved in interactive marketing.

An interactive channel is a tool 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 only have one set of production runtime and staging servers, or to divide your events and interaction points by customer facing system.

Interactive flowcharts

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. In the case of 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, as well as 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 a variety of ways.

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

Offers are re-usable:

- 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 target cells in flowcharts using one of the contact processes, and track your campaign results by capturing data about customers who received the offer, and customers who responded.

Profiles

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

The runtime environment uses the information 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 web sites 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).

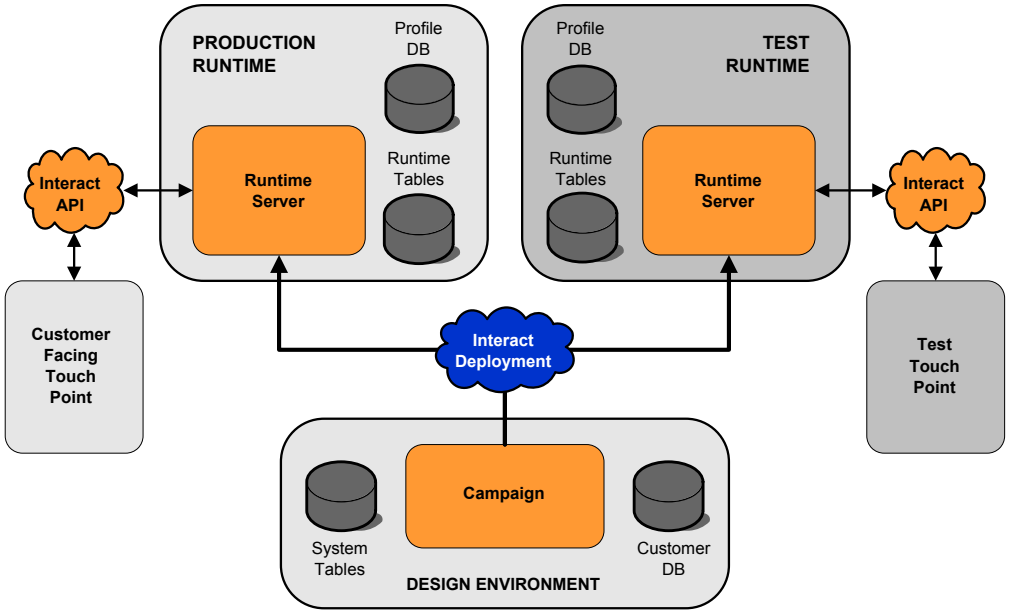
Treatment rules

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 may 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.
- If you have multiple server groups using a distributed `cacheType`, each should use a unique multicast port. Using both a unique multicast port and address for each server group is better.
- 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.

Chapter 2. Configuring IBM Unica Interact 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 installing Interact, you must configure at least one Interact user, the runtime environment user.

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

Important: All runtime servers belonging to the same server group must share the same user credentials. If you have separate Marketing Platform instances for each runtime server, you must create the same user and password on each.

If you are using a database load utility, you must define the runtime tables as a data source with login credentials for the runtime environment user. The name of this data source must be `systemTablesDataSource`.

If you enable security for JMX monitoring with the JMXMP protocol, you may need a separate user for JMX monitoring security.

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.

You should grant any Campaign user who has permission to edit interactive flowcharts access to the test run tables data source.

If you have Interact installed and configured, the following additional options are available for the default Global Policy and new policies. Remember that some design environment users also require some Campaign permissions such as Custom Macros.

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

For example, you can create two roles, one for the people who create interactive flowcharts, and one for the people who define the interaction strategies. Each section lists the permissions granted the role.

Interactive flowchart role

Custom Macro

- Add Custom Macros
- Edit Custom Macros
- Use Custom Macros

Derived Field

- Add Derived Fields
- Edit Derived Fields
- Use Derived Fields

Flowchart Template

- Paste Templates

Segment Template

- Add Segments
- Edit Segments

Session

- 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

Campaign

- 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

- View Offer Summary

Segment Template

- View Segment Summary

Session

- 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.

Working with Interact data sources

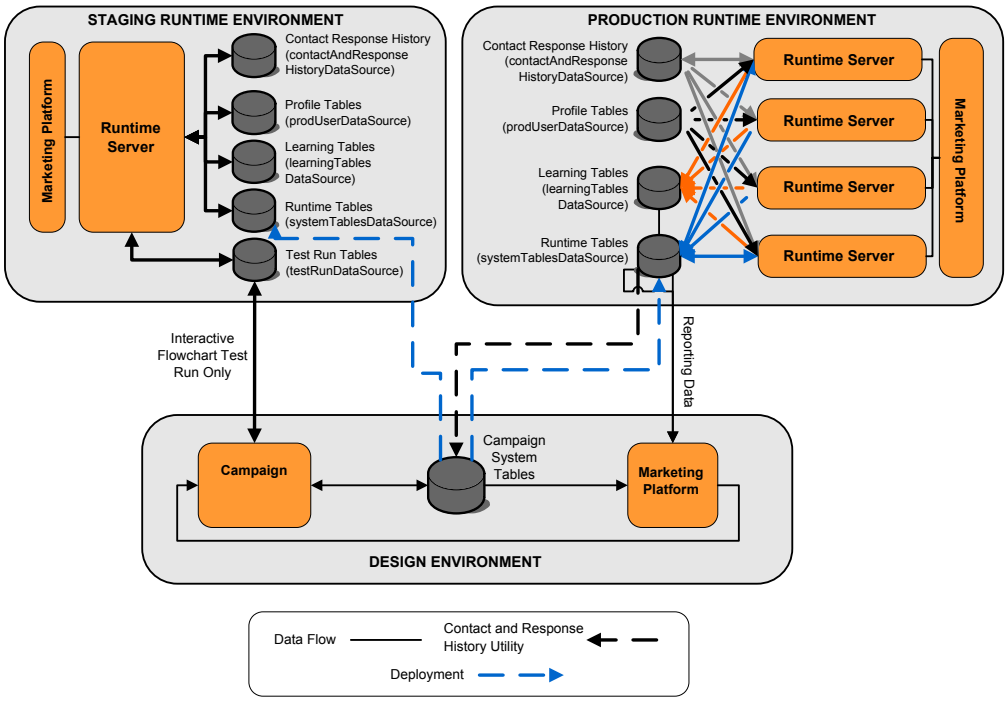
Interact requires several sets of data to function.

- **Campaign system tables** — beyond all the data for Campaign, the Campaign system tables contain data for Interact components you create in the design environment, such as treatment rules and interactive channels. The design environment and the Campaign system tables share the same physical database and schema.
- **Runtime tables** — (`systemTablesDataSource`) contains the deployment data from the design environment, staging tables for contact and response history, and runtime statistics.
- **Profile tables** — (`prodUserDataSource`) contains any customer data—beyond information gathered in real time—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 used by the interactive channel.

  The profile tables can also contain the tables used for augmenting offer serving, including tables for offer suppression, score override, and global and individual offer assignment.
- **Test run tables** — (`testRunDataSource`) contains a sample of all data 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 designated as the test run server group for the design environment only.
- **Learning tables** — (`learningTablesDataSource`) contains all data gathered by the built-in learning utility. These tables can include a table which defines dynamic attributes. If you are not using learning or are using an external learning utility you create, you do not need learning tables.
- **Contact and response history for cross-session response** — (`contactAndResponseHistoryDataSource`) 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 following diagram shows the possible Interact data sources and how they relate to the IBM® Unica® 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 design environment, you also create new, Interact-specific tables in the Campaign system tables.

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 will use 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.

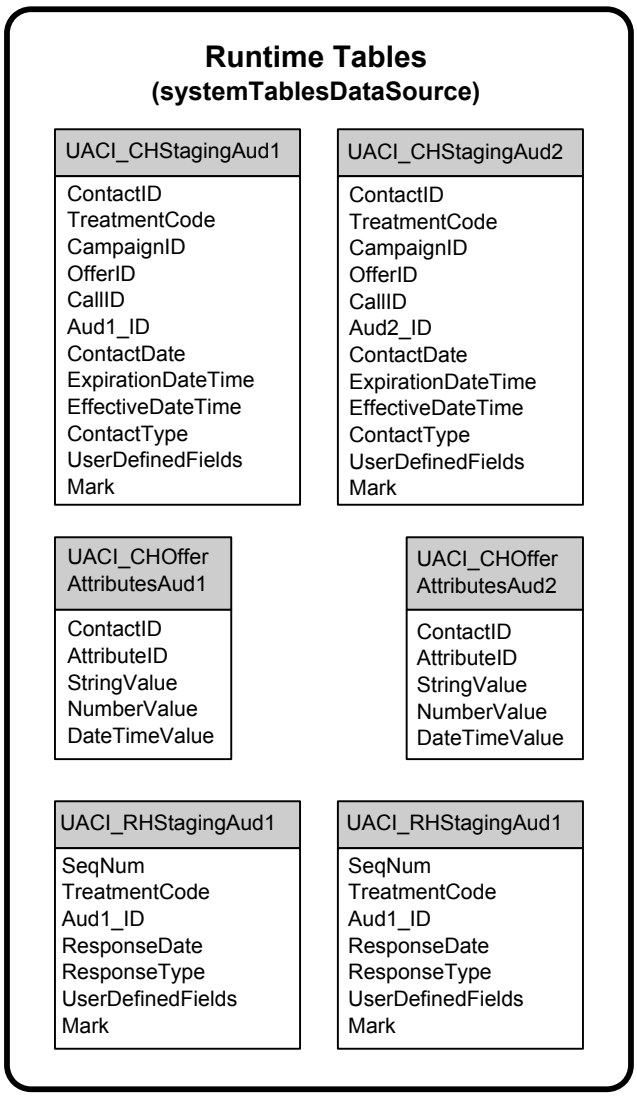
The SQL scripts create the following tables for the default audience level:

- 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 testDataSource, or if the prodUserDataSource is of a different database type, Interact then checks the prodUserDataSource for the table.

To override the default data types

Follow these steps to override the default data types for dynamically created tables.

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

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

   Important: The runtime server must be restarted whenever changes are made to the uaci_column_types table.

Default data types for dynamically created tables

The following table lists the hardcoded data types that the Interact runtime system uses by default for each supported database 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_usertab` 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_usertab` 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.

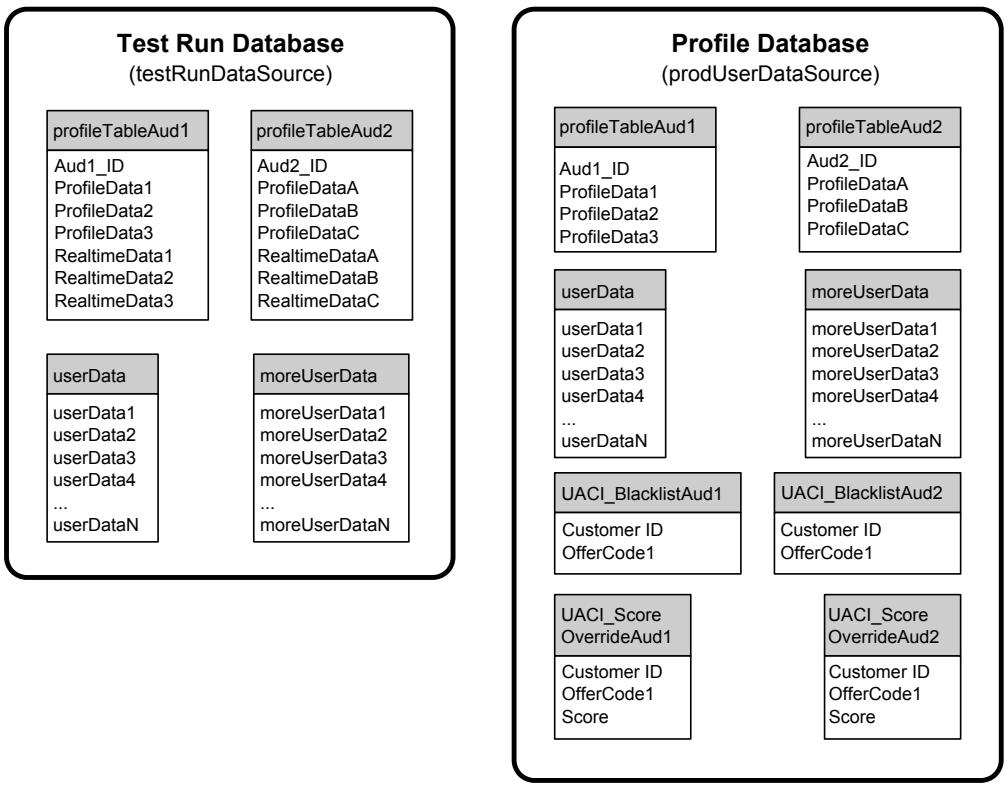
  The offer suppression table can contain a row for each offer suppressed for an audience member, although an entry is not required for all audience members. You can create a sample offer suppression table, `UACI_BlackList` by running the `aci_usertab` SQL script against your profile database.

  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_usertab` 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_OfferAllTx`. 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.

Using Interact feature scripts

Several of the optional features available with Interact require changes to specific tables in your profile databases. Your Interact installation, both design environment and runtime environment, includes feature ddl scripts. These scripts add any specific columns required to your table.

To enable any of these features, run the appropriate script against the appropriate database or table.

`dbType` is the database type, for example `sqlsvr` for Microsoft SQL Server.

Feature Name	Feature Script	Run Against	Change
Global offers, offer suppression, and score override	Runtime environment installation directory\ddl\ `aci_usrtab_`*`dbType`*`.sql`	Your profile database (userProdDataSource)	Creates the `DefaultOffers`, `UACI_BlackList`, and `UACI_ScoreOverride` tables.
Scoring	Runtime environment installation directory\ddl\ `acifeatures\ aci_scoringfeature_`*`dbType`*`.sql`	Score override tables in your profile database (userProdDataSource)	Adds the `LikelihoodScore` and `AdjExploreScore` columns.
Learning	Design environment installation directory\ddl\acifeatures\ `aci_lrnfeature_`*`dbType`*`.sql`	Campaign database containing contact history tables	Adds the column `RTSelectionMethod` to the `UA_DtlContactHist` table.

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 `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.

Configuring contact and response types

You can record one contact type and two response types with Interact as shown in the following table. All of these properties 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`

You can also record additional custom response types using the `postEvent` method.

You should also 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 you must define a value for all 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.

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.

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** — A number with the value 1 or 0. 1 indicates Interact should log the event as an accept for learning. 0 indicates Interact should not log the event for learning. This parameter enables you to create several `postEvent` methods logging different response types without influencing learning. If you do not define `UACILOGTOLEARNING`, Interact assumes the default value of 0.

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 following tables show how the runtime environment staging tables map to the Campaign history tables. Remember that you should have one of these tables for each audience level. The table names shown are the sample tables created for the default audience in the runtime tables and the Campaign system tables.

Table 2. 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
UserDefinedFields	UA_DtlContactHist	UserDefinedFields

`ContactID` is a key to join the `UACI_CHOfferAtrib` with `UACI_CHStaging`.

Table 3. 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
DateTimeValue	UA_OfferHistAttrib	DateTimeValue

ContactID is a key to join the UACI_CHOfferAtrib with UACI_CHStaging.

Table 4. 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 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 you choose. 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 will copy that data to the linkFrom column in the UA_DtlContactHist table.

Important: If you have any additional columns in your Campaign contact and response history tables, you must 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 column with the same names as your name-value pairs in your runtime session data. For example, if you create name-value pairs NumberItemsInWishList and NumberItemsInShoppingCart, when a Log Offer Acceptance or Log Offer Rejection event occurs, if the columns NumberItemsInWishList and NumberItemsInShoppingCart exist in your UACI_RHStaging table, the runtime environment populates those fields. The runtime environment populates the UACI_CHStaging table when a Log Offer Contact event occurs.

You can use these user defined fields to include the score used to present an offer. Add a column 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 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.

To configure JMX monitoring for the contact and response history module

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	**JMXMP** or **RMI** Security is not enabled for the contact and response history module, even if you select the JMXMP protocol.

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

The default address for monitoring the contact and response history module with the JMXMP protocol is `service:jmx:jmxmp://`*`CampaignServer`*`:`*`port`*`/campaign`.

The default address for monitoring the contact and response history module with the RMI protocol is `service:jmx:rmi:///jndi/rmi://`*`CampaignServer`*`:`*`port`*`/campaign`.

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.

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.

1. The contact and response history tables must be mapped properly in Campaign.
2. You must 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.
3. If you are going to copy the contact history tables to the runtime environment, do so now.
4. 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.
5. Create a version of the UACI_XsessResponse table for each audience level.

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.

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.

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.
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

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

Column	Type	Description
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 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.
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 you want to include when 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.

To enable cross-session 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. This 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.

Current State	Description
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.

To enable a database load utility with 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 the runtime environment user has login credentials for the runtime tables data source defined in Marketing Platform.

   The name of the data source in Marketing Platform must be `systemTablesDataSource`.
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.

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.

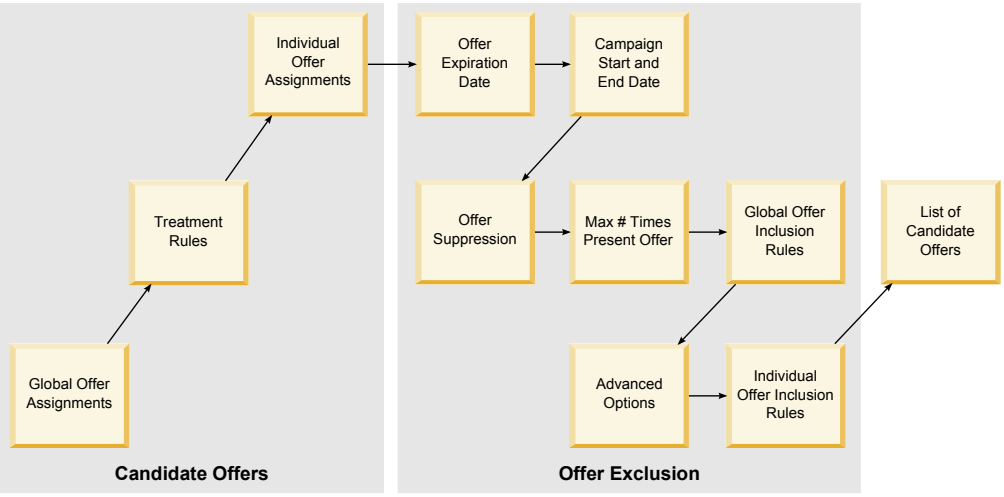
Treatment rules are only the start of how Interact determines which offers are eligible for a customer. Interact has several optional features which you can implement to enhance how the runtime environment determines which offers to present. None of these features guarantee that an offer is presented 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.

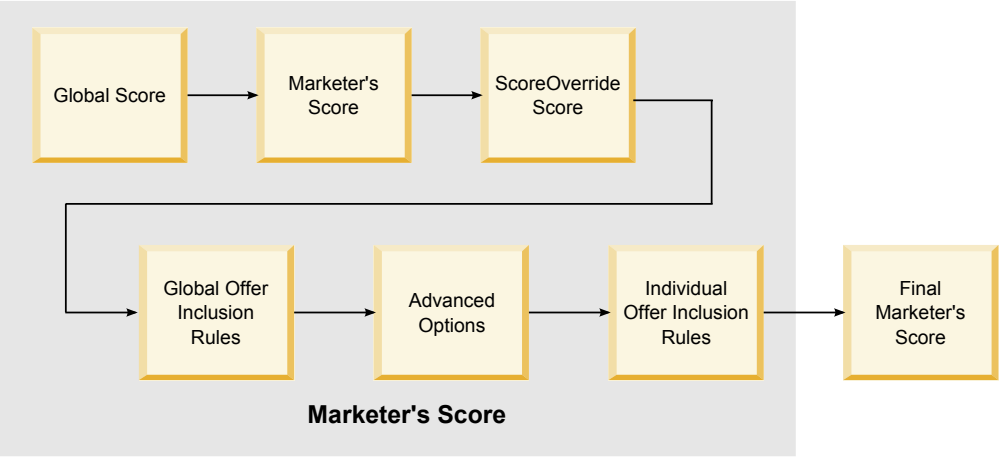


- **Global offer assignments** — You can define global offers by audience level using the global offers table.

- **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.

Calculating the marketing score

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

About suppressing 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 for the duration of the session.

To enable the offer suppression table

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

1. Create an `offerSuppressionTable`, a new table for every audience containing the audience ID and the offer ID.
2. Set the `enableOfferSuppressionLookup` property to **true**.
3. Set the `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_usertab` SQL script against your profile database. The `aci_usertab` 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.

To define 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 by defining the `DefaultCellCode` property for each audience level and table type in the `IndividualTreatment` category.

The `DefaultCellCode` is used if you have not defined a 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 defined in Campaign. This cell code is used for all offer assignments appearing in reporting. If you define unique default cell codes, you can easily identify offers assigned by the default offers or score override tables.

To define the UACI_ICBatchOffers table

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 not used in a treatment rule must be defined in the `UACI_ICBatchOffers` table.

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

You must populate the `UACI_ICBatchOffers` table with offer codes used in the default offer or score override tables. The table takes 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, if required.
OfferCode3	varchar(64)	The third part of the offer code, if required.
OfferCode4	varchar(64)	The fourth part of the offer code, if required.
OfferCode5	varchar(64)	The fifth part of the offer code, if required.

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

To enable the global offer table

You can configure the runtime environment to assign global offers for an audience level, beyond anything defined in treatment rules.

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

   You can create the `UACI_DefaultOffers` table with the correct columns using the `aci_usrtab` ddl file.
2. Set the `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_usertab` SQL script against your profile database. The `aci_usertab` 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 an 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

To enable score override table

You can configure Interact to use a score 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.

   You can create a sample score override table with the correct columns using the `aci_usrtab` ddl file.
2. Set the `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_usertab SQL script against your profile database. The aci_usertab 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_usertab 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.

Understanding Interact learning

The Interact learning module monitors visitor's responses to offers and visitor attributes. The learning module has two general modes:

- **Exploration**—the learning module serves offers in order to gather enough response data to optimize the estimation used later during exploitation. Offers served during exploration do not necessarily reflect the optimal choice.
- **Exploitation**—after enough data has been collected by the exploration phase, the learning module uses the probabilities to help select which offers to present.

The learning module alternates between exploration and exploitation based on two properties: a confidence level you configure with the `confidenceLevel` property

and a probability that the learning module presents a random offer you configure with the percentRandomSelection property.

You set the confidenceLevel to a percentage which 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 has been presented as many times as defined by the minPresentCountThreshold, the learning module enters the exploration mode. Without a lot of 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 as well as historical acceptance data and the current session data. The formula inherently balances between exploration and exploitation, and returns the appropriate weight.

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 forces the learning module to recommend offers other than the most successful to determine if other offers would be more successful if they had greater exposure. For example, if you configure percentRandomSelection to 5, this means that 5% of the time, the learning module presents a random offer and adds the response data to its calculations.

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

1. Calculates the probability a visitor will select an offer.
2. Calculates the offer weight using the probability from step 1 and determines whether it should be in exploration or exploitation mode.
3. Calculates a final score for each offer using the marketing score and the offer weight from step 2.
4. Sorts the offers by the scores 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 that it should 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.

Learning is also based on the recencyWeightingFactor property and the recencyWeightingPeriod property. These properties enable you to add more weight to more recent data than older data. The recencyWeightingFactor is the percentage of weight the recent data should have. The recencyWeightingPeriod is the length of time that is recent. For example, you configure the recencyWeightingFactor to .30 and the recencyWeightingPeriod to 24. This means that the previous 24 hours of data are 30% of all data considered. If you have 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.

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.

To enable the learning module

All runtime servers have a built-in learning module. By default, this learning module is disabled. You can enable learning 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 an attribute stored in a dimension table you reference in an interactive flowchart, or some event parameter you collect in real time.

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.
- 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 `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.

To define a learning attribute

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

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

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

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

To 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 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 5. MyProfileTable

VisitorID	KeyField
1	Key1
2	Key2
3	Key3
4	Key4

Table 6. 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 7. UACI_AttributeList

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

To enable external learning

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.

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.

You must 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.

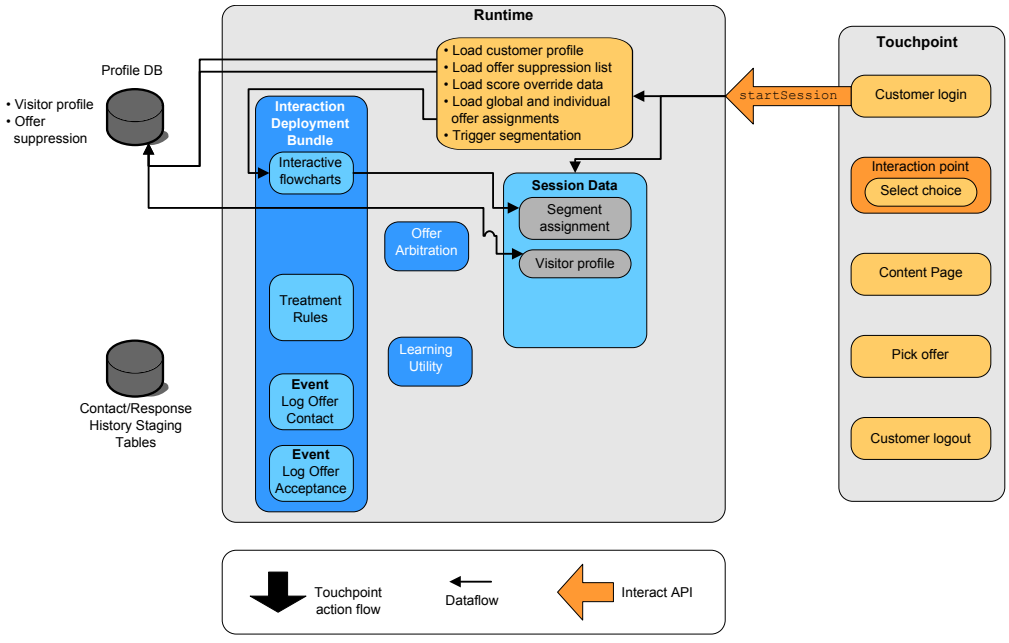
The Interact application programming interface (API) allows you to configure your touchpoint and a runtime server to work together to serve the best possible offers. Using the API, the touchpoint can request information from the runtime server to assign the visitor to a group (a segment) and present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

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

The following figure shows a simple implementation of the Interact API. A visitor logs into a web site and navigates to a page which displays offers. The visitor selects an offer and logs out. While the interaction is simple, there are several events which occur both in the touchpoint and the runtime server.

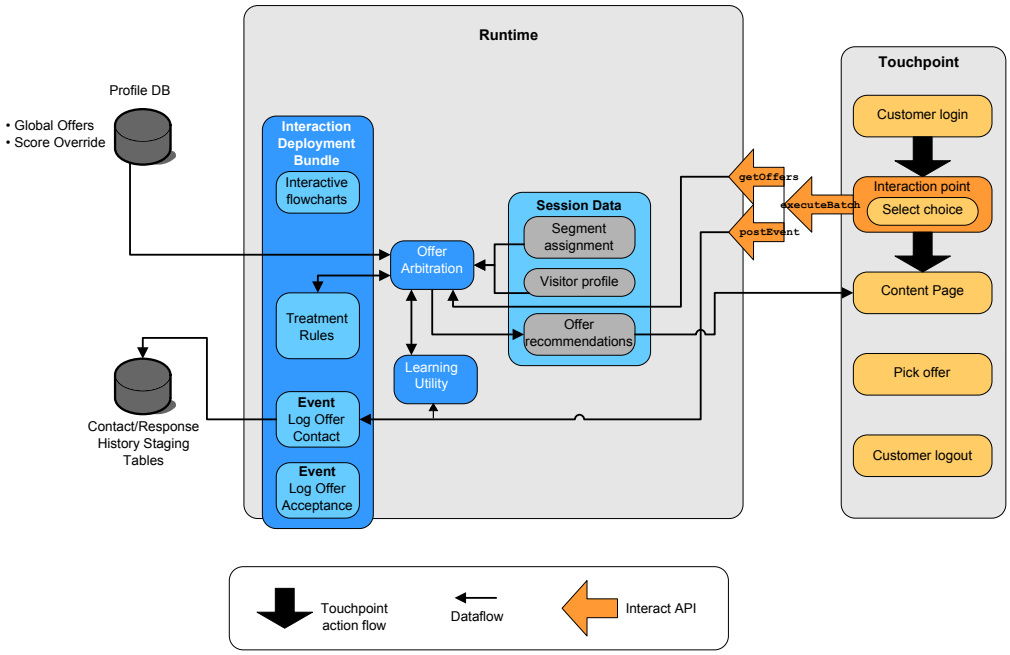
When a visitor logs in, this triggers a `startSession`.



In this example, the `startSession` method does four things. First, it creates a new runtime session. Second, it sends a request to load the customer profile data into the session. Third, 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.

Fourth, the runtime loads any offer suppression and global and individual offer treatment information into the session. The session data is held in memory for the duration of the session.

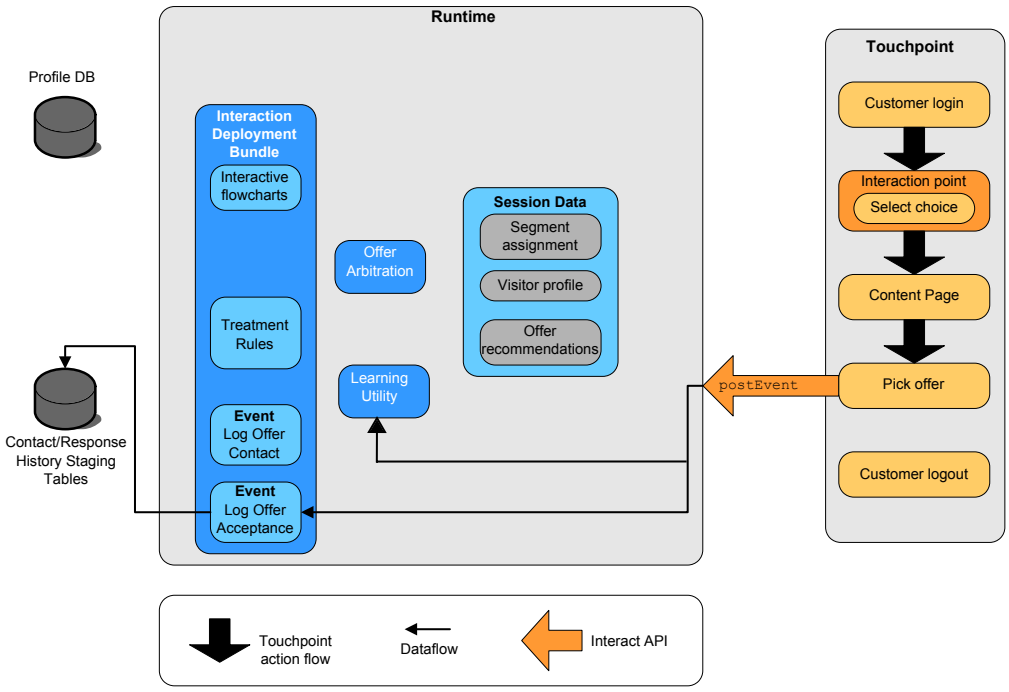
The visitor navigates the site until the visitor reaches a pre-defined interaction point. In the figure, the second interaction point (Select choice) is a place where the visitor clicks a link that presents a set of offers. The touchpoint manager configured the link to trigger an `executeBatch` method.



The `executeBatch` method enables you to 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 uses the segmentation data, the offer suppression list, the treatment rules, and the learning module to propose a set of offers. The runtime returns a set of offers which are displayed on the content page.

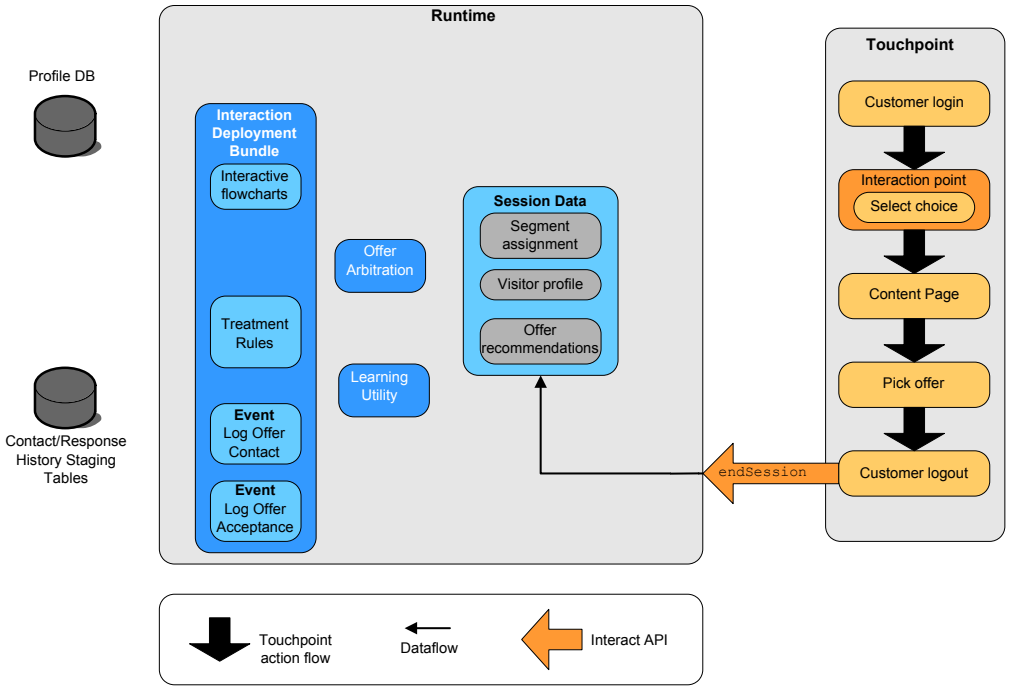
The `postEvent` method triggers one of the events defined in the design environment. In this particular case, the event sends a request to log the offers presented to contact history.

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



The button 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.

The visitor, after selecting the offer, is finished with the web site and logs out. The log out command is linked to the `endSession` method.



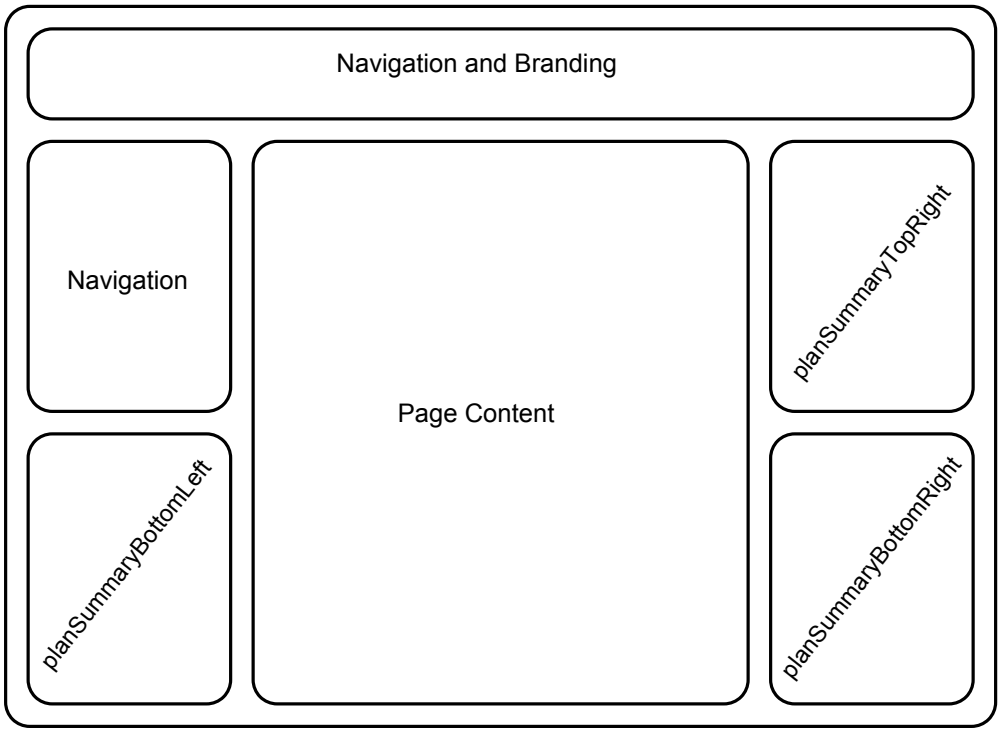
The `endSession` method closes the session. If the visitor forgets to log out, there is a configurable session timeout to ensure that all sessions eventually end. If you want to keep any of the data passed to the session, such as information included in parameters in the `startSession` or `setAudience` 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 containing the Snapshot process.

This example is very simple (the visitor only takes four actions — log in, navigate to page that displays offers, select an offer, and log out — which is a simple interaction) to show the basics of how the API works between your touchpoint and the runtime environment. You can design your integration to be as complicated as you need (within the limits of your performance requirements).

Simple interaction planning example

You are designing an interaction for a cellular phone company's website. 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.

One offer to be displayed in a zone that is dedicated to offers about upgrades

- The area on the page displaying the upgrade offer must be defined. Also, once 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 displaying 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 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. 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);
```

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 will display an offer, you need to first get an offer, then extract the information 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 returns a response object including an OfferList response. However, your web page can not 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()+ " >   */
  }
 }
}
```

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 because the Interact the Interact runtime server automatically logs the last recommended list of treatments as contacts if you do not supply the UACIOfferTrackingCode.

You have also written something to change the image displayed every 30 seconds for the second area on the page displaying a phone upgrade offer. You have decided to rotate between three images, so should you should 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 has been displayed. To log the contact, the UACITrackingCode parameter needs to be posted as before. Each offer will have 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 should log the offer accepted and the offers 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.

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

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

Remember the following tips as you write your integration.

- When designing your touchpoint, create some default filler content (usually a benign branding message or empty content) for every interaction point where offers can be presented. This is in case no offers are eligible to be served to the current visitor in the current situation. You should assign this default filler content as the default string for the interaction point.

- When designing your touchpoint, include some method of presenting content in case your touchpoint cannot reach the runtime server group for some unforeseen reason.
- When triggering events that re-segment your visitor, including `postEvent` and `setAudience`, remember that running flowcharts does take some amount of time. The `getOffers` method waits until segmentation is finished before executing. Overly frequent re-segmentation may hinder `getOffers` call response performance.
- You need to decide what an "offer rejection" means. Several reports, such as the Channel Offer Performance Summary report, present the number of times an offer has been rejected. This is a count of the number of times the Log Offer Rejection action has been triggered by a `postEvent`. You need to determine whether Log Offer Rejection is for an actual rejection (such as clicking a link labeled "No, thanks.") or for an offer that is ignored (such as a page displaying three different banner ads, none of which are selected).
- There are several optional features you can enable to enhance Interact offer selection, including learning, offer suppression, individual offer assignments, and 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 Unica 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.

To configure Interact to use JMX monitoring with the RMI protocol

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

Configuration property	Setting
enableSecurity	**False** The Interact implementation of the RMI protocol does not support security.

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

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.

Note: 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.

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

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

Using the jconsole scripts

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

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 hostname of the runtime server you want to monitor.
4. Set PORT to the port you have configured JMX to listen on with the Interact > monitoring > port property.
5. If you are monitoring via the RMI protocol, add a comment before the JMXMP connection and remove the comment before the RMI connection.

   The script monitors via the JMXMP protocol by default.

   For example, see the default settings for jconsole.bat.

   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
```

JMX attributes

The following tables describe the attributes available with JMX monitoring.

All data provided by JMX monitoring is since last reset or system start. For example, a count is of the number of items since last reset or system start, not since installation.

Table 8. 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 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 written to the response history table.
ErrorCount	The number of error messages 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 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.

Table 8. Contact Response History ETL Monitor (continued)

Attribute	Description
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 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 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 currently running.
NumberOfResponseHistoryRecordsMarked	The number of response history records in the `UACI_RHStaging` table 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 currently running.

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. Exceptions

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

Table 10. Flowchart Engine Statistics

Attribute	Description
activeProcessBoxThreads	Active count of flowchart process threads (shared between all executions) that are currently running.

Table 10. Flowchart Engine Statistics (continued)

Attribute	Description
activeSchedulerThreads	Active count of Flowchart Scheduler threads that are currently running.
avgExecutionTimeMillis	Average flowchart execution time in milliseconds.
CurrentJobsInProcessBoxQueue	The number of jobs waiting to be run by flowchart process threads.
CurrentJobsInSchedulerQueue	The number of jobs 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 have completed.
numExecutionsStarted	Total number of flowchart executions started.

Table 11. Specific flowcharts by interactive channel

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

Table 12. Locale

Attribute	Description
locale	Locale setting for JMX client.

Table 13. Logger Configuration

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

Table 14. Services Thread Pool Statistics

Attribute	Description
activeContactHistThreads	The approximate number of threads that are actively executing tasks for Contact History and Response History.
activeFlushCacheToDBThreads	The approximate number of threads that are actively executing tasks to flush cached statistics to the data store.
activeOtherStatsThreads	The approximate number of threads that are actively executing tasks for Eligible Stats, Event Activities and Default Stats.
CurrentHighWaterMarkInContactHistQueue	Greatest number of entries queued to be logged by the service which collects the contact and response history data.
CurrentHighWaterMark InFlushCachetoDBQueue	Greatest number of entries queued to be logged by the service which writes the data in the cache to the database tables.
CurrentHighWaterMarkInOtherStatsQueue	Greatest number of entries queued to be logged by the service which 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 used for Contact History and Response History.
currentMsgsInFlushCacheToDBQueue	The number of jobs in the queue for the thread pool used to flush cached statistics to the data store.
currentMsgsInOtherStatsQueue	The number of jobs in the queue for the thread pool used for Eligible Stats, Event Activities, and Default Stats.
maximumContactHistThreads	The largest number of threads that have ever simultaneously been in the pool used for Contact History and Response History.
maximumFlushCacheToDBThreads	The largest number of threads that have ever simultaneously been in the pool used for flushing cached statistics to the data store.
maximumOtherStatsThreads	The largest number of threads that have ever simultaneously been in the pool used for Eligible Stats, Event Activities and Default Stats.

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 which 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 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 15. Service Statistics

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

Table 16. 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 recreating new ones.

Table 16. Service Statistics - Database Load Utility (continued)

Attribute	Description
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 created.
NumberOfLoaderStagingDirRemoved	The total number of staging directories removed.
NumberOfLoaderStagingDirMovedToAttention	The total number of staging directories renamed to attention.
NumberOfLoaderStagingDirMovedToError	The total number of staging directories 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 retrieving the primary key from database.
NumberOfLoaderThreadsRuns	The total number of database loader threads runs.
NumberOfFlushCacheFiles	The total number of times flushing file cache.

Table 17. API Statistics

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

Table 17. API Statistics (continued)

Attribute	Description
LoadOffersBySQLCount	The number of `LoadOffersBySQL` API calls since last reset or system start.
LoadOffersBySQLDuration	Time elapsed for the last `LoadOffersBySQL` API call.
loadProfileDuration	Time elapsed for the last `loadProfile` API call.
loadScoreOverrideDuration	Time elapsed for the last `loadScoreOverride` API call.
postEventCount	The number of `postEvent` API calls since last reset or system start.
postEventDuration	Time elapsed for the last `postEvent` API call.
runSegmentationDuration	Time elapsed for the last `runSegmentation` API call.
setAudienceCount	The number of `setAudience` API calls since last reset or system start.
setAudienceDuration	Time elapsed for the last `setAudience` API call.
setDebugCount	The number of `setDebug` API calls since last reset or system start.
setDebugDuration	Time elapsed for the last `setDebug` API call.
startSessionCount	The number of `startSession` API calls since last reset or system start.
startSessionDuration	Time elapsed for the last `startSession` API call.

Table 18. Learning Optimizer Statistics

Attribute	Description
LearningOptimizerAcceptCalls	The number of accept events passed into the learning module.
LearningOptimizerAcceptTrackingDuration	The total number of milliseconds spent logging the accept events in the learning module.
LearningOptimizerContactCalls	The number of contact events passed into the learning module.
LearningOptimizerContactTrackingDuration	The total number of milliseconds spent logging the contact events in the learning module.
LearningOptimizerLogOtherCalls	The number of non-contact and non-accept events passed into the learning module.
LearningOptimizerLogOtherTrackingDuration	The duration in milliseconds spent in logging other events (non-contact and non-accept) in the learning module.
LearningOptimizerNonRandomCalls	The number of times the configured learning implementation was applied.

Table 18. Learning Optimizer Statistics (continued)

Attribute	Description
LearningOptimizerRandomCalls	The number of times the configured learning implementation was bypassed and random selection was applied.
LearningOptimizerRecommendCalls	The number of recommend requests passed into the learning module.
LearningOptimizerRecommendDuration	The total number of milliseconds spent in the learning recommend logic.

Table 19. Default Offer Statistics

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

JMX operations

The following table describes the operations available for JMX monitoring.

Group	Attribute	Description
Logger Configuration	activateDebug	Set log level for the log file defined in `Interact/conf/interact_log4j.properties` to debug.
Logger Configuration	activateError	Set log level for the log file defined in `Interact/conf/interact_log4j.properties` to error.
Logger Configuration	activateFatal	Set log level for the log file defined in `Interact/conf/interact_log4j.properties` to fatal.
Logger Configuration	activateInfo	Set log level for the log file defined in `Interact/conf/interact_log4j.properties` to info.
Logger Configuration	activateTrace	Set log level for the log file defined in `Interact/conf/interact_log4j.properties` to trace.
Logger Configuration	activateWarn	Set log level for the log file 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.
Default Offer Statistics	updatePollPeriod	Updates defaultOfferUpdatePollPeriod. This value, in seconds, tells the system how long to wait before updating the default offers in the cache. If set to `-1`, the system only reads the number of default offers at start up.

Chapter 7. Classes and methods for the IBM Unica Interact 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.

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

1. Before you can start working with the Java serialization adapter, you must add the following file to your `CLASSPATH`:

   `Interact_Runtime_Environment_Installation_Directory/lib/interact_client.jar`
2. All objects passed back and forth between the client and the server can be found in the package `com.unicacorp.interact.api`. See the Interact API JavaDoc for more details.
3. To get an instance of the InteractAPI class, call the static method `getInstance` with the url of the Interact runtime server.

SOAP prerequisites

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 using SOAP, you must do the following:

1. Convert the Interact API WSDL using the SOAP toolkit of your choice.
   The Interact API WSDL is installed with Interact in the `Interact/conf` directory.
   The text of the WSDL is available at the end of this guide.
2. Install and configure the runtime server.
   The runtime server must be running to fully test your integration.

SOAP Versions

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 Web site:

Apache Axis2

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

API JavaDoc

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

About API examples

All of the examples in this guide were created using the Java serialization over HTTP adapter. If you are using SOAP, because the classes generated from the WSDL can vary based on the SOAP toolkit and the options you select, these examples may not work exactly 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

`endSession(String sessionID)`

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.

- **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

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

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

- **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());
  }
}
```

getInstance

```
getInstance(String URL)
```

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

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

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

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

- **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

`getOffersForMultipleInteractionPoints(String sessionID, String requestStr)`

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

- **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

`getProfile(String sessionID)`

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

- **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

```
getVersion()
```

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

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

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

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

- **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.

Unless specified with the UACIExecuteFlowchartByName parameter, re-segmentation runs all interactive flowcharts associated with this interactive channel for the current audience level. 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 may 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

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

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

- **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

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

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

- **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

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

The startSession method creates and defines a runtime session. 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());
}
```

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 for learning. 0 indicates the runtime environment should not log the event for learning. 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).

Feature	Parameter	Description
	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.
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

```
getDetailMessage()
```

The `getDetailMessage` method returns the detailed, verbose description of an Advisory Message object.

Not all messages have a detailed message.

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

```
getMessage()
```

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

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

```
getMessageCode()
```

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

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

```
getStatusLevel()
```

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

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

Code	Description
1	INVALID_SESSION_ID - the session ID does not reference a valid session
2	ERROR_TRYING_TO_ABORT_SEGMENTATION - encountered an error while trying to abort segmentation during an `endSession`
3	INVALID_INTERACTIVE_CHANNEL - the argument passed in for interactive channel does not reference a valid interactive channel
4	INVALID_EVENT_NAME - the argument passed in for the event does not reference a valid event for the current interactive channel
5	INVALID_INTERACTION_POINT - the argument 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
10	COMMAND_METHOD_UNRECOGNIZED - command method specified for a command within an `executeBatch` 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 lookup the interaction point name
17	RUNTIME_EXCEPTION_ENCOUNTERED - an unexpected runtime exception was encountered
18	ERROR_TRYING_TO_EXECUTE_ASSOCIATED_ACTION - error trying to execute associated action (Trigger Re-segmentation, Log Offer Contact, Log Offer Acceptance, or Log Offer Rejection)
19	ERROR_TRYING_RUN_FLOWCHART - error trying to run flowchart
20	FLOWCHART_FAILED - flowchart failed
21	FLOWCHART_ABORTED - flowchart aborted
22	FLOWCHART_NEVER_RUN - flowchart never run
23	FLOWCHART_STILL_RUNNING - flowchart is still running
24	ERROR_WHILE_READING_PARAMETERS - error while reading parameters
25	ERROR_WHILE_LOADING_RECOMMENDED_OFFERS - error while loading recommended offers

Code	Description
26	ERROR_WHILE_LOGGING_DEFAULT_TEXT_STATISTICS - error while logging default text statistics (the number of times the default string for the interaction point displayed)
27	SCORE_OVERRIDE_LOAD_FAILED - loading the score override table failed
28	NULL_AUDIENCE_ID - audience ID is empty
29	UNRECOGNIZED_AUDIENCE_LEVEL - unrecognized audience level
30	MISSING_AUDIENCE_FIELD - audience field missing
31	INVALID_AUDIENCE_FIELD_TYPE - invalid audience field type
32	UNSUPPORTED_AUDIENCE_FIELD_TYPE - unsupported audience field type
33	TIMEOUT_REACHED_ON_GET_OFFERS_CALL - getOffers call reached timeout without returning offers
34	INTERACT_INITIALIZATION_NOT_COMPLETED_SUCCESSFULLY - runtime initialization did not complete successfully
35	SESSION_ID_UNDEFINED - session ID undefined
36	INVALID_NUMBER_OF_OFFERS_REQUESTED - invalid number of offers requested
37	NO_SESSION_EXIST_BUT_WILL_CREATE_NEW_ONE
38	AUDIENCE_ID_NOT_FOUND_IN_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
41	AUDIENCE_NOT_DEFINED_IN_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

```
getBatchStatusCode()
```

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

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

```
getResponses()
```

The getResponses method returns the array of response objects that correspond to the array of commands executed by the executeBatch method.

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_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

`setAudienceID(`*`audienceID`*`)`

The `setAudienceID` method defines the AudienceID for the `setAudience` and `startSession` commands.

- **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

`setAudienceLevel(`*audienceLevel*`)`

The `setAudienceLevel` method defines the Audience Level for the `setAudience` and `startSession` commands.

- *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

`setDebug(`*debug*`)`

The `setDebug` method defines the debug level for the `startSession` command. 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

`setEvent(`*event*`)`

The `setEvent` method defines the name of the event used by the `postEvent` command.

- **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

`setEventParameters(`*`eventParameters`*`)`

The `setEventParameters` method defines the event parameters used by the `postEvent` command. These values are stored in the session data.

- **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

```
setGetOfferRequests(numberRequested)
```

The **setGetOfferRequests** method sets the parameter for retrieving offers used by the getOffersForMultipleInteractionPoints command.

- **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

`setInteractiveChannel(`*`interactiveChannel`*`)`

The `setInteractiveChannel` method defines the name of the interactive channel used by the `startSession` command.

- **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

`setInteractionPoint(`*`interactionPoint`*`)`

The `setInteractionPoint` method defines the name of the interaction point used by the `getOffers` and `postEvent` commands.

- **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

`setMethodIdentifier(`*`methodIdentifier`*`)`

The `setMethodIdentifier` method defines the type of command contained in the command object.

- **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

setNumberRequested(*numberRequested*)

The setNumberRequested method defines the number of offers requested by the getOffers command.

- **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

`setRelyOnExistingSession(relyOnExistingSession)`

The `setRelyOnExistingSession` method defines a boolean defining whether the `startSession` command uses an existing session or not.

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

`getName()`

The `getName` method returns the name component of a NameValuePair object.

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

`getValueAsDate()`

The `getValueAsDate` method returns the value of a NameValuePair object.

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

```
getValueAsNumeric()
```

The getValueAsNumeric method returns the value of a NameValuePair object.

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

```
getValueAsString()
```

The getValueAsString method returns the value of a NameValuePair object.

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

```
getValueDataType()
```

The getValueDataType method returns the data type of a NameValuePair object.

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

```
setName(name)
```

The setName method defines the name component of a NameValuePair object.

- **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

`setValueAsDate(`*valueAsDate*`)`

The `setValueAsDate` method defines the value of a NameValuePair object.

- **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

`setValueAsNumeric(`*valueAsNumeric*`)`

The `setValueAsNumeric` method defines the value of a NameValuePair object.

- **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

`setValueAsString(`*valueAsString*`)`

The `setValueAsString` method defines the value of a NameValuePair object.

- **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

`getValueDataType(`*`valueDataType`*`)`

The `setValueDataType` method defines the data type of a NameValuePair object.

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

```
getAdditionalAttributes()
```

The `getAdditionalAttributes` method returns the custom offer attributes defined in Campaign.

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

```
getDescription()
```

The `getDescription` method returns the description of the offer defined in Campaign.

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

```
getOfferCode()
```

The `getOfferCode` method returns the offer code of the offer as defined in Campaign.

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

```
getOfferName()
```

The `getOfferName` method returns the name of the offer as defined in Campaign.

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

```
getScore()
```

The `getScore` method returns one of the following:

- If you have 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 have 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 have 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 representing 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

```
getTreatmentCode()
```

The `getTreatmentCode` method returns the treatment code of the offer as defined in Campaign.

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

```
getDefaultString()
```

The getDefaultString method returns the default string for the interaction point as defined in Campaign.

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

```
getRecommendedOffers()
```

The getRecommendedOffers method returns an array of Offer objects requested by the getOffers method.

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

```
getAdvisoryMessages()
```

The getAdvisoryMessages method returns an array of Advisory Messages from the Response object.

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

```
getApiVersion()
```

The getApiVersion method returns the API version of a Response object.

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

```
getOfferList()
```

The `getOfferList` method returns the OfferList object of a Response object.

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

```
getAllOfferLists()
```

The `getAllOfferLists` method returns an array of all OfferLists of a Response object.

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

```
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.

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

```
getSessionID()
```

The getSessionID method returns session ID.

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

```
getStatusCode()
```

The getStatusCode method returns the status code of a Response object.

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. 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 Unica 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

To add a web service for use with EXTERNALCALLOUT

The EXTERNALCALLOUT macro recognizes callouts only if you have 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 new 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

```
getNumberOfArguments()
```

The getNumberOfArguments method returns the number of arguments expected by the Java class with which you are integrating.

Return value

The getNumberOfArguments method returns an integer.

Example

The following example shows printing the number of arguments.

```
public int getNumberOfArguments()
{
    return 0;
}
```

getValue

```
getValue(audienceID, configData, arguments)
```

The getValue method performs the core functionality of the callout and returns the results.

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

```
initialize(configData)
```

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.

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

```
shutdown(configData)
```

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.

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

1. Create a file called GetCreditScore.java with the following contents. This file assumes there is a class 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 your

You must compile your implementation against `interact_externalcallout.jar` located in the `lib` directory of your IBM Unica 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`.

To add a data source for use with Profile Data Services

The `EXTERNALCALLOUT` macro recognizes a data source for Profile Data Services hierarchical data import only if you have 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 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`
`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 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.)

Chapter 9. IBM Unica 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 Unica 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

##############################################################################
```

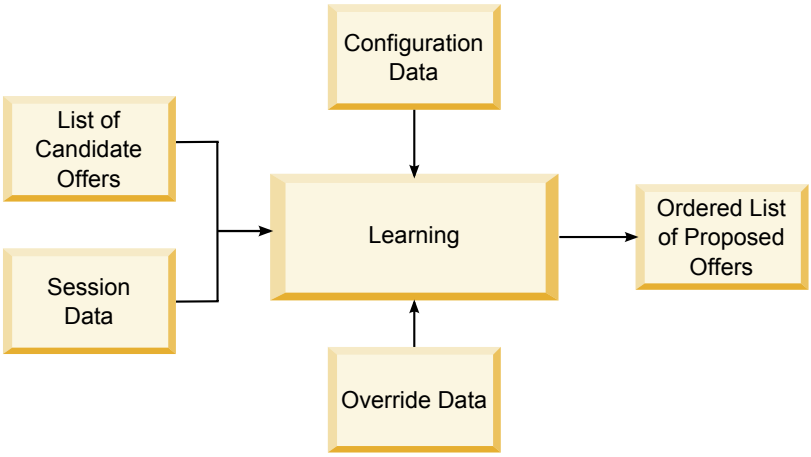
```
# dtPW:  this is the PW that goes along with the dtLogin
################################################################################
dtPW=

################################################################################
# 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 10. 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.

To enable external learning

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.

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.

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

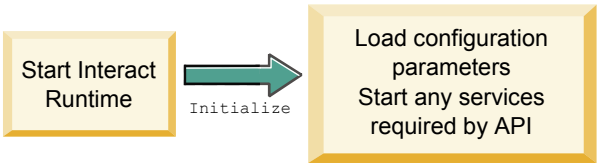
ILearning interface

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

```
initialize(ILearningConfig config, boolean debug)
```



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.

- **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.

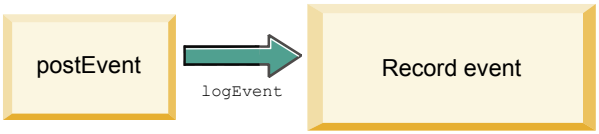
If the `initialize` method fails for any reason, it throws an `LearningException`.

Return value

None.

logEvent

```
logEvent(ILearningContext context,
             IOffer offer,
             IClientArgs clientArgs,
             IInteractSession session,
             boolean debug)
```



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.

- **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.

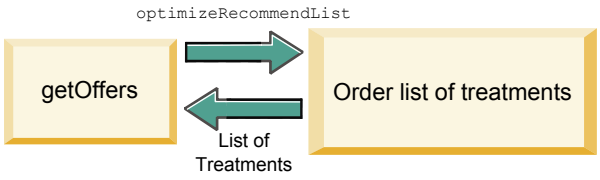
If the `logEvent` method fails, it throws a `LearningException`.

Return value

None.

optimizeRecommendList

```
optimizeRecommendList(list(ITreatment) recList,
  IClientArgs clientArg, IInteractSession session,
  boolean debug)
```



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.

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.

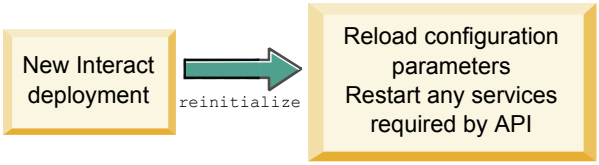
If the `optimizeRecommendList` method fails, it throws a `LearningException`.

Return value

The `optimizeRecommendList` method returns a List of `ITreatment` objects.

reinitialize

```
reinitialize(ILearningConfig config,
                  boolean debug)
```



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.

- **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.

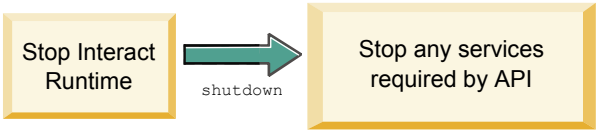
If the `logEvent` method fails, it throws a `LearningException`.

Return value

None.

shutdown

`shutdown(ILearningConfig` *`config`*`, boolean` *`debug`*`)`



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.

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

```
getAudienceLevel()
```

The getAudienceLevel method returns audience level.

Return value

The getAudienceLevel method returns a string that defines the audience level.

getComponentNames

```
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.

Return value

A set of strings containing the names of the components of the audience ID.

getComponentValue

```
getComponentValue(String componentName)
```

The getComponentValue method returns the value for the named component.

- **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

```
getValue(int clientArgKey)
```

The getValue method returns the value of the requested map element.

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

```
getAudienceId()
```

The `getAudienceId` method returns an `AudienceID` object. Use the `IAudienceID` interface to extract the values.

Return value

The `getAudienceId` method returns an `AudienceID` object.

getSessionData

```
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.

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

```
getDataType(string parameterName)
```

The `getDataType` method returns the data type for the specified parameter name.

Return value

The `getDataType` method returns an InteractDataType object. IntearctDataType is a Java enum represented by Unknown, String, Double, Date, or List.

getParameterNames

```
getParameterNames()
```

The getParameterNames method returns a set of all the names of the data in the current session.

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

```
getValue(parameterName)
```

The getValue method returns the object value corresponding to the specified parameterName. Object can either be a String, Double, or a Date.

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

```
setValue(string parameterName, object value)
```

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.

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

```
getName()
```

The getName method returns the name of the learning attribute.

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

```
getLearningContext()
```

The getLearningContext method return the constant that tells us whether or not this is a contact, accept or reject scenario.

- **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

```
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.

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

```
getCreateDate()
```

The getCreateDate method returns the date the offer was created.

Return value

The getCreateDate method returns a date that defines the date the offer was created.

getEffectiveDateFlag

```
getEffectiveDateFlag()
```

The getEffectiveDateFlag method returns a number that defines the effective date of the offer.

- **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

```
getExpirationDateFlag()
```

The getExpirationDateFlag method returns an integer value that describes the expiration date of the offer.

- **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

```
getOfferAttributes()
```

The getOfferAttributes method returns offer attributes defined for the offer as an IOfferAttributes object.

Return value

The `getOfferAttributes` method returns an `IOfferAttributes` object.

getOfferCode

```
getOfferCode()
```

The `getOfferCode` method returns the offer code of the offer as defined in Campaign.

Return value

The `getOfferCode` method returns an `IOfferCode`object.

getOfferDescription

```
getOfferDescription()
```

The `getOfferDescription` method returns the description of the offer defined in Campaign.

Return value

The `getOfferDescription` method returns a string.

getOfferID

```
getOfferID()
```

The `getOfferID` method returns the offer ID as defined in Campaign.

Return value

The `getOfferID` method returns a long that defines the offer ID.

getOfferName

```
getOfferName()
```

The `getOfferName` method returns the name of the offer as defined in Campaign.

Return value

The `getOfferName` method returns a string.

getUpdateDate

```
getUpdateDate()
```

The `getUpdateDate` method returns date of when the offer was last updated.

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

```
getParameterNames()
```

The getParameterNames method returns a list of the offer parameter names.

Return value

The getParameterNames method returns a set that defines the list of offer parameter names.

getValue

```
getValue(String parameterName)
```

The getValue method returns value of the given offer attribute.

Return value

The getValue method returns an object that defines the value of the offer attribute.

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

```
getPartCount()
```

The getPartCount method returns the number of parts that make up an offer code.

Return value

The getPartCount method returns an integer defining the number of parts of the offer code.

getParts

```
getParts()
```

The getParts method gets an unmodifiable list of the offer code parts.

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

```
getOfferCode()
```

The getOfferCode method returns the value of the offer code columns in the score override table for this audience member.

Return value

The getOfferCode method returns an IOfferCode object that defines the value of the offer code columns in the score override table.

getParameterNames

```
getParameterNames()
```

The getParameterNames method returns the list of parameters.

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

```
getValue(String parameterName)
```

The `getValue` method returns the value of the zone column in the score override table for this audience member.

- **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

```
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.

Return value

The `getCellCode` method returns a string that defines the cell code.

getCellId

```
getOfferName()
```

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.

Return value

The getCellId method returns a long that defines the cell ID.

getCellName

```
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.

Return value

The getCellName method returns a string that defines the cell name.

getLearningScore

```
getLearningScore()
```

The getLearningScore method returns the score for this treatment. 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

```
getMarketerScore()
```

The getMarketerScore method returns the marketer's score defined by the slider on the interaction strategy tab for the offer.

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

```
getOffer()
```

The getOffer method returns the offer for the treatment.

Return value

The `getOffer` method returns an `IOffer` object that defines the offer for this treatment.

getOverrideValues

```
getOverrideValues()
```

The `getOverrideValues` method returns overrides defined in the default offers or score override table.

Return value

The `getOverrideValues` method returns an `IScoreOverride` object.

getPredicate

```
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.

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

```
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.

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

```
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 representing the score of the offer.

getTreatmentCode

```
getTreatmentCode()
```

The `getTreatmentCode` method returns the treatment code.

Return value

The `getTreatmentCode` method returns a string that defines the treatment code.

setActualValueUsed

```
setActualValueUsed(string parmName, object value)
```

Use the `setActualValueUsed` method to define what values are used at various stages in the learning algorithm execution.

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);

        }
    }

}
```

Appendix A. IBM Unica Interact WSDL

```
<?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>
```

Appendix B. 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 Unica Marketing 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

No default value defined.

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.

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.
<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.

Token	Description
<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 Unica 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 Unica 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 | 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 `Unica > 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 Unica > 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 Unica > 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.

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, even if enableOfferSuppressionLookup is set to true.

Interact looks for this table in the customer tables accessed by runtime servers, defined by the prodUserDataSource properties.

Default value

UACI_BlackList

profileTable

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 name of the table containing the profile data for this audience level.

Interact looks for this table in the customer tables accessed by 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_usrProd. If you enter a fully-qualified name, for example, mySchema.UACI_usrProd, Interact does not prepend the schema name.

Default value

No default value defined.

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.

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.

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.

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.

Default value

UACI_XSessResponse

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 Unica | 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

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`, Interact, 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.

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 baysean 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.

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.

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

aggregateStatsIntervalInMinutes

Description

The number of minutes the Interact learning module waits between aggregating data in 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

15

Valid Values

An integer greater than zero.

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 | 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 | 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 | 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 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 | 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 | 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 | sessionManagement

This set of configuration properties defines settings for runtime sessions.

cacheType

Description

Defines the type of cache approach for the runtime servers.

Default value

`Local`

Valid Values

Distributed | Local

maxNumberOfSessions

Description

The maximum number of runtime sessions that the cache holds at any one time. If a request to add a new runtime session occurs when the cache has reached this maximum, the cache removes the oldest inactive runtime session.

Default value

999999999

Valid Values

Integer greater than 0.

multicastIPAddress

Description

If `cacheType` is `Distributed`, enter the IP address used by the distributed cache. You must also define `multicastPort`.

If `cacheType` is `Local`, you can leave `multicastIPAddress` undefined.

Default value

230.0.0.1

Valid Values

Any valid IP address.

multicastPort

Description

If cacheType is Distributed, enter the port number used by the distributed cache. You must also define multicastIPAddress.

If cacheType is Local, you can leave multicastPort undefined.

Default value

6363

Valid Values

1024 – 49151

sessionTimeoutInSecs

Description

The amount of time, in seconds, a session can remain inactive. Once the sessionTimeout number of seconds have passed, Interact ends the session.

Default value

300

Valid Values

Any integer greater than zero.

Appendix C. Interact design environment configuration properties

This section describes all the configuration properties for Interact design environment.

Campaign | partitions | partition[n] | reports

These configuration properties define 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 using 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 using 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 using 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 using 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 using 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 using 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 using 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 using 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 have installed 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 have installed 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, four arguments are passed to the completion notification script: start time, end time, total number of CH records processed, and total number of RH records processed. The start time and end time are numeric values representing number of milliseconds elapsed since 1970.

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

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.

enableLearning

Description

If set to Yes, the Interact design time expects learning to be enabled. If you set enableLearning to yes, you must configure Interact > offerserving > optimizationType to BuiltInLearning or ExternalLearning.

If set to No, the Interact design time expects learning to be disabled. If you set enableLearning to no, you must configure Interact > offerserving > optimizationType to NoLearning.

Default value

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.

Default value

5

Valid Values

Any integer from 0 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.

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

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

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

`4294967295`

eMessageInstalled

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 have installed eMessage, you must configure this property manually.

Default value

`no`

Valid Values

`yes | no`

interactInstalled

Description

After installing the Interact design environment, this configuration property should be set to `yes` to enable the Interact design environment in Campaign.

If you do not have Interact 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 have installed Interact.

MO_UC_integration

Description

Enables integration with Marketing Operations for this partition. If you plan to set any of the following three options to Yes, you must set **MO_UC_integration** to Yes. For more information about configuring this integration, see the *IBM Unica Marketing Operations and Campaign Integration Guide*.

Default value

no

Valid Values

yes | no

MO_UC_BottomUpTargetCells

Description

Allows bottom-up cells for Target Cell Spreadsheets on this partition. When set to Yes, both top-down and bottom-up target cells are visible, but bottom-up target cells are read only. Note that **MO_UC_integration** must be enabled. For more information about configuring this integration, see the *IBM Unica Marketing Operations and Campaign Integration Guide*.

Default value

no

Valid Values

yes | no

Legacy_campaigns

Description

When the **MO_UC_integration** property is set to **Yes**, the **Legacy_campaigns** property enables access to campaigns created before enabling integration, including campaigns created in Campaign 7.x and linked to Plan 7.x projects. For more information about configuring this integration, see the *IBM Unica Marketing Operations and Campaign Integration Guide*.

Default value

no

Valid Values

yes | no

IBM Unica Marketing Operations - Offer integration

Description

Enables the ability to use Marketing Operations to perform offer lifecycle management tasks on this partition. (**MO_UC_integration** must be enabled. Also, **Campaign integration** must be enabled in **Settings > Configuration > Unica > Platform**.) For more information about configuring this integration, see the *IBM Unica Marketing Operations and Campaign Integration Guide*.

Default value

no

Valid Values

yes | no

UC_CM_integration

Description

Enables IBM Coremetrics® online segment integration for a Campaign partition. If you set this option to yes, the Select process box in a flowchart will provide the option to select **IBM Coremetrics Segments** as input. To configure the integration for each partition, choose **Settings > Configuration > Campaign | partitions | partition[n] | Coremetrics**.

Default value

no

Valid Values

yes | no

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

yes

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 yes, enables Campaign JMX connector server for Interact. Campaign has no JMX security.

If set to no, 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.

Appendix D. 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 Unica 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 218. 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 Unica 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 Unica 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 214.
- Installing the Message Connector web application as described in "Deploying and running the Message Connector" on page 215.
- 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 216.

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 20. 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 21. 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 22. 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 23. 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 24. 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 Unica 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 Unica 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 (where <*Interact_home*> is the directory into which you installed the Interact runtime, such as C:\Unica\Interact or /Unica/Interact). The following table lists the sample SQL scripts you can use to manually create the Message Connector tables:

Table 25. 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 26. 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 Unica 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 Unica 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 214

Just as you deploy other IBM Unica 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, such as C:\Unica\Interact, or /Unica/Interact.

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 Unica 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 Unica 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_installation_directory>/jsconnector.

   For example, on a Windows installation, this might be C:\Unica\Interact\jsconnector. On a UNIX system, you might enter /Unica/Interact/jsconnector for this value.

   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 Unica 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 Unica 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 directory, such as `C:\Unica\Interact` or `/Unica/Interact`.
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 221 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.

   For example, on a Windows installation, this might be `C:\Unica\Interact\jsconnector`. On a UNIX system, you might enter `/Unica/Interact/jsconnector` for this value.

   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.
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 220.

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 27. 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 223 for details.

Table 27. Web Connector Configuration Settings Summary (continued).

Section	Settings
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 225 for details.
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 227 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), NetInsight 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 Unica 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 28. 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>*

Table 28. Site-wide settings for the Web Connector installation (continued)

Setting	Description	Equivalent setting in jsconnector.xml
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 29. 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 NetInsight Page Tag Integration	Specifies (with a `yes` or `no` answer) whether the Web Connector should attach the specified IBM Unica NetInsight tag at the end of the page content.	`<enableNetInsightTagging>`
NetInsight Tag HTML Template File	The HTML/Javascript template used to integrate a call to the NetInsight 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 Unica 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 30. 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 30. 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 30. 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 NetInsight 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 NetInsight 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 31. 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.
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.

Table 31. Web Connector Admin Page Options (continued)

Option	Purpose
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 testPage.html has been included with the Interact Web Connector 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: Unica Interact Web Connector Page Tag -->
<!-- Copyright 2011, IBM Corporation All rights reserved. -->
<script language="javascript" type="text/javascript">
//<![CDATA[
   var unicaWebConnectorBaseURL = "http://localhost:7001/interact/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: Unica Interact Web Connector Page Tag -->
   <!--
################################################################################
end of pageTag paste
################################################################################
-->
   </body>
</html>
```

Appendix E. Interact and Intelligent Offer Integrated Product Recommendations

IBM Unica Interact can integrate with IBM Coremetrics Intelligent Offer to provide Interact-driven product recommendations. Both products can provide product recommendations for offers, but using different methods. Intelligent Offer 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 Intelligent Offer 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 Intelligent Offer

This section describes how IBM Unica Interact can integrate with IBM Coremetrics Intelligent Offer to provide Interact-driven product recommendations, including a description of the process, and the mechanisms by which the integration takes place.

IBM Unica Interact integrates with IBM Coremetrics Intelligent Offer via a Representational state transfer (REST) application programming interface (API), made available from the Intelligent Offer 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 Intelligent Offer 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 Intelligent Offer 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 Intelligent Offer, and get cutlery product recommendations.

Integration Prerequisites

Before you can use the Intelligent Offer - 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 Intelligent Offer REST API as described in your Intelligent Offer developer documentation.
- You have a basic understanding of HTML, JavaScript, CSS, and JSON (JavaScript Object Notation).

  JSON is important because the Intelligent Offer 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 Intelligent Offer 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 Intelligent Offer REST API link (a URL for your Intelligent Offer 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 239 for more information.

Configuring an offer for Intelligent Offer integration

Before your web page can call IBM Coremetrics Intelligent Offer to retrieve a recommended product, you must first configure the IBM Unica Interact offer with the necessary information to pass to Intelligent Offer.

To set up an offer to link to Intelligent Offer, 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 Intelligent Offer 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 Intelligent Offer, perform the following steps.

1. Create or edit an offer for Interact.

   For information on creating and modifying offers, see the *IBM Unica Interact User's Guide*, and the IBM Unica 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 Intelligent Offer category ID associated with this offer.

     You can retrieve the category ID manually from your Intelligent Offer 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 Intelligent Offer 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 Intelligent Offer integration. The remaining work to allow Intelligent Offer 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 Intelligent Offer 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 Intelligent Offer - 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 Intelligent Offer 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
#  Unica 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 IntelligentOffer.
 *
 * 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 IntelligentOffer 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 : IntelligentOffer 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 Intelligent Offers 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 IO 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
 *   Intelligent Offer
 *****************************************************************************/

 /*****************************************************************************
 * Call IntelligentOffer 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());
   }
%>
```

Contacting IBM Unica 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 Unica technical support. Use the information in this section to ensure that your problem is resolved efficiently and successfully.

If you are not a designated support contact at your company, contact your IBM Unica administrator for information.

Information to gather

Before you contact IBM Unica technical support, gather the following information:

- A brief description of the nature of your issue.
- Detailed error messages 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 Unica 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 Unica applications.

You can access the About page by selecting **Help > About**. If the About page is not accessible, you can obtain the version number of any IBM Unica application by viewing the `version.txt` file located under the installation directory for each application.

Contact information for IBM Unica technical support

For ways to contact IBM Unica technical support, see the IBM Unica Product Technical Support website: (http://www.unica.com/about/product-technical-support.htm).

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 about 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.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites 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
170 Tracer Lane
Waltham, MA 02451
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.

Printed in USA