PureConnect®

2021 R2

Generated:

05-May-2021

Content last updated:

26-September-2019

See Change Log for summary of changes.

Interaction Web Tools

Developer's Guide

Abstract

This document describes how to customize the Interaction Web Tools user interface and its features.

For the latest version of this document, see the PureConnect Documentation Library at: http://help.genesys.com/pureconnect.

Table of Contents

Introduction to Interaction Web Tools

Interaction Web Tools enables users to provide web-based chat and callback interactions to their customers.

By default, the feature displays a webpage with three tabs: for starting a chat, for requesting a callback, and for creating an account.

The mobile versions of Web Tools are responsive, meaning that they adjust to the size and shape of the screen on a mobile device. On the Android device at the left, the **Username** and **Password** labels are above the text boxes. On the iOS device at the right, the labels are to the left because the device has a larger screen.
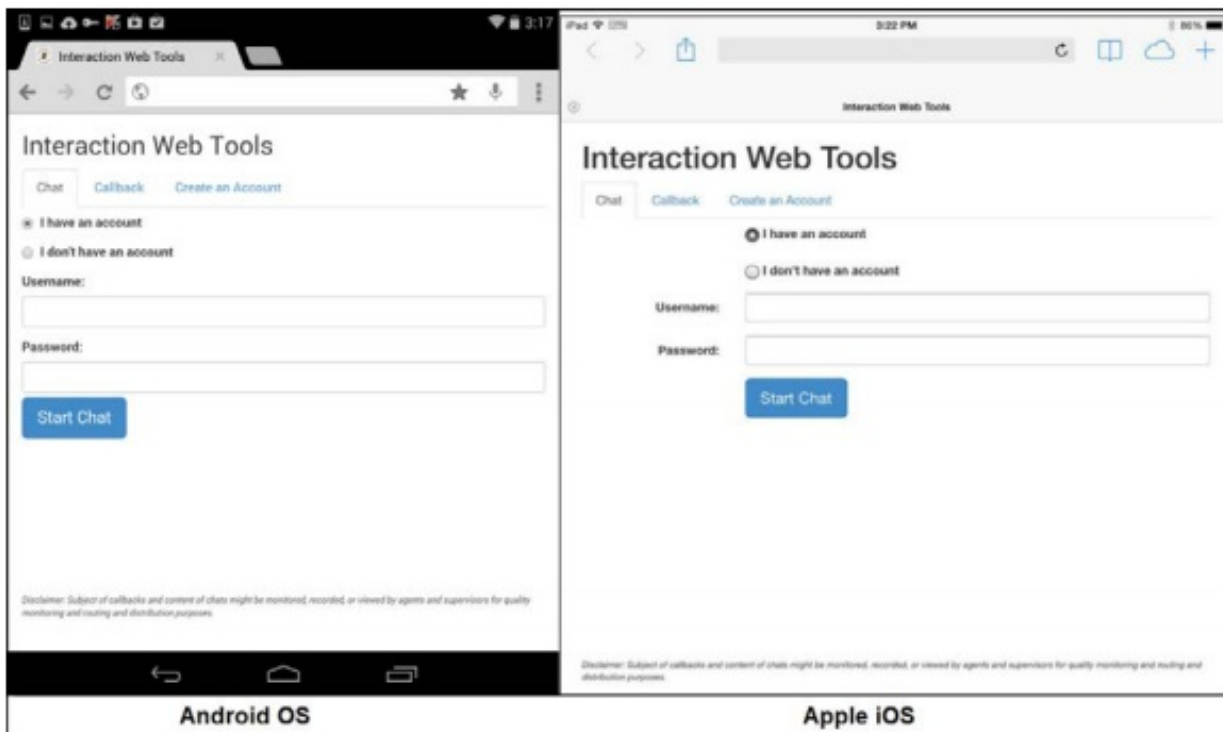
By using the features of Interaction Web Tools, you can modify the page in various ways. For example:

- Change color, background, or CSS styles.
- Embed the page within another webpage.
- Add controls to the page.
- Create a customized client interface, still sending and receiving messages in PureConnect formats.
- Change the chat or callback window that appears to the user.

Customizing Interaction Web Tools features requires knowledge of JavaScript, JSON, and related technologies in order to use the API.

Chat Transcripts

If an administrator enables the chat transcript feature, website visitors can choose to have a transcript of the chat emailed to themselves. The transcript feature is available if:

- The CIC server has an Interaction Recorder license, and
- The EnableWebTranscript web services parameter (set in Interaction Administrator) has a non-zero value.

For more information about enabling chat transcripts, see "Enable Chat Transcripts" in the *Interaction Web Tools Technical Reference* at https://help.genesys.com/cic/mergedProjects/wh_tr/desktop/pdfs/web_services_tr.pdf.

General Information about Protocols

Here are some general points about using the API:

- All URIs start with "/WebSvcs" from the WebProcessorBridge's perspective. However, from the web server's perspective, other URI components probably appear first and trigger the reverse proxy mechanism.
- A guid represents the notion of <chat, participant> or <callback, participant>. Guids are present at the end of most URIs. That does not include URIs for querying the server's capabilities, starting a new chat, and creating a callback, however. When these URIs are called, there is no <chat, participant> to specify.
- You can write all request/response pairs in JSON. The language is controlled by an HTTP Accept header in the requests and the Content type in the responses. This is the same regardless of whether you are using a GET or a POST. The correct MIME type to use is:

  `application/json`

- You can optionally use a semicolon and a character encoding, for example:

  `text/json;charset=utf-8`

- Most JSON requests encountered during an active chat receive a response in the common JSON response format. See [Common Response Format](#) for details on these response formats.
- The following types of requests do not get a common response:
  - A request for a server's configuration
  - A request to create a callback
  - A request for the status of a callback
  - A request to cancel a callback
  - A request to query the queue
  - A request for party information
  - The successful response to a request to retrieve a file sent by the agent is the contents of that file. However, the unsuccessful response is an HTTP 500 error.
- The client uses the following URI components. The URI components are dynamically created, based on the following information:
  - Server name
  - URI components to trigger the reverse proxy
  - Path info, which is hard-coded into the client
  - Guid, which is received from the server
- All URIs are case-insensitive.

With switchover, the web server is essentially a single point of failure and there are two different reverse proxies, one for each CIC server.

For example, suppose that the web server is chat.company.com and the CIC servers are server1.example.com and server2.example.com. The administrator sets up the reverse proxy so that http://chat.company.com/server1/path goes to:

`http://server1.company.com:8114/path`

Likewise, http://chat.company.com/server2/path goes to:

`http://server2.company.com:8114/path`

If a chat starts on …/server1/… and then a switchover occurs from server1 to server2, Interaction Web Tools tells them to start using …/server2/… URIs (which goes to server2).

Making Customizations

`I3Root/styles` includes several style sheets. You can also write your own style sheets and link to them from `index.html`.

You can change some behavior by changing Interaction Web Tools parameters in `index.html`. You can also extend the functionality of Interaction Web Tools by adding subclasses in `customizations.js`.

Customization examples

`IWT_Examples-X-X.zip` contains multiple examples, which include styles, JavaScript customization examples, and a readme file for each example. This document uses the name of the folder in `IWT_Examples-X-X.zip` to refer to individual examples.

For example, "the RegistrationFormPanelFields example" refers to `IWT_Examples-X-X.zip/RegistrationFormPanelFields`.

The examples use the same files and folder structure as the default installation. In some cases, two index files are present when an example requires users to select options.

> **Note**: The location of the customization files depends on the selections the system administrator made when installing Interaction Web Tools.

To use the examples

1. Make a backup of `I3Root/js/config.js`.
2. Make a backup of `I3Root`.
3. Copy the contents of the example folder to `I3Root`.
4. Place the backed-up copy of `config.js` back in `I3Root/js`.

This process overwrites any existing customizations you have made. To avoid overwriting existing customizations, you can alternatively copy code from the examples to the corresponding files in `I3Root`.

Interaction Web Tools parameters

The `index.html` file contains the `setInteractionWebToolsParams` function, which defines configuration settings for Interaction Web Tools. You can modify the values of these parameters to change the behavior of Interaction Web Tools.

By default, several of the parameters are commented out. You can uncomment them to add functionality.

The following table describes the Interaction Web Tools parameters.

> **Note:**
> You can edit `config.js` by hand or by using `IWT_ConfigUtility.exe`.

Parameter	Default Value	Description
currentUriFragment	ui.ConfigConversions.convertICServerCountToCurrentUriFragment(config.ICServerCount)	The part of the URI the web server recognizes as a request to be reverse proxied to the CIC server. Defaults to the value in `config.js`, which is `I3Root/Server1` if you follow the default installation instructions.
uriFragments	ui.ConfigConversions.convertICServerCountToUriFragments(config.ICServerCount)	The set of possible URI fragments used to reverse proxy to the CIC server. Defaults to the value in `config.js`.

Parameter	Default Value	Description
pageMode	ui.ConfigConversions.convertInteractionTypesToPageMode(config.InteractionTypes)	The types of interactions that can be handled. Defaults to a numeric representation of the interaction types set in `config.js`, i.e. 1 for chats and 2 for callbacks.
chatTarget	config.ChatTarget	The user or workgroup to send chats to. Defaults to the value in `config.js`.
chatTargetType	config.ChatTargetType	The chat target type: user or workgroup.
callbackTarget	config.CallbackTarget	The user or workgroup to send callback requests to. Defaults to the value in `config.js`
callbackTargetType	config.CallbackTargetType	The callback target type: user or workgroup.
defaultLanguageCode	config.DefaultLanguageCode	The default language. Defaults to the value in `config.js`.
customInfo	null	Information that is displayed in the chat window in the CIC client. Only agents see this information.
chatFollowupUrl	null	A URL to redirect to when a chat finishes.
callbackAttributes	null	Extra attributes that are added to a callback request.
callbackRoutingContext	null	Category/context pairs that you can use to route callback requests based on skills. See Implementing Routing Contexts.
chatRoutingContext	null	Category/context pairs that you can use to route chats based on skills. See Implementing Routing Contexts.

Parameter	Default Value	Description
useHttps	config.UseEncryption	Whether to user HTTP or HTTPS. Defaults to the value in `config.js.`
urlsAreRelative	false	Whether to use relative URLs when building urls for Web Tools functionality. By default, Interaction Web Tools prepends URLs with http://server:port/ using the server information set in `config.js.` Using relative URLs instead allows you to set up the reverse proxies up somewhere other than `I3Root/Server1` and `I3Root/Server2.`

Using customizations.js

Customizations.js includes several customization points that allow you to override default functionality with your own classes. WebServices.js includes documentation of classes and their parameters.

See the examples, such as RegistrationFormPanelFields, for information about extending a class.

Change default send on enter behavior

The web chat interface includes a check box indicating whether chats are sent when a user pushes the Enter key.
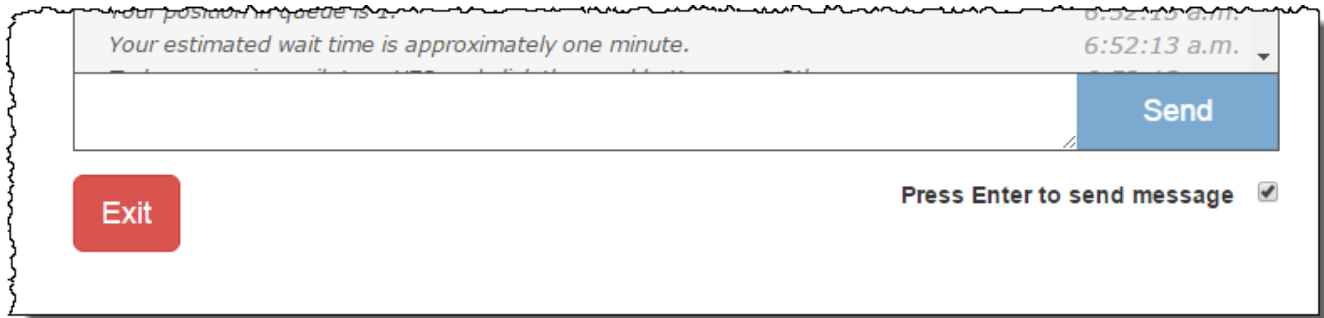
The check box defaults to **off** for users of Arabic, Hebrew, Japanese, Korean, Russian, Serbian, Turkish, and Chinese. It defaults to **on** for all other languages.

You can change the default behavior of this check box by using the SendOnEnter customization point in customizations.js. Create a subclass of webservices._Internal._DefaultSendOnEnter and override the getSendOnEnterByDefault method.

The following example changes the default to off in all cases:

```
customizations.singletonImplementations.SendOnEnter =
 Class.create(webservices._Internal._DefaultSendOnEnter,
  {
    getSendOnEnterByDefault : function()
      {
        return false;
      }
  }
);
```

Chats and Callbacks to Multiple Queues

In the default installation:

- Interaction Web Tools directs all chats to a single user or workgroup queue (or chats are unavailable).
- Interaction Web Tools directs all callbacks to a single user or workgroup queue (or callbacks are unavailable).
- The chat queue and the callback queue needn't be the same.

However, your organization can allow chats and callbacks to multiple queues. For example, suppose you want a directory page with links to chat to individual employees. Another example is a site with multiple products that has a callback link on each product page directed to a queue for that product.

Implementing Routing Contexts

For more information about implementing routing contexts, see *Interaction Web Tools Technical Reference*.

To use skill-based routing:

1. Create and assign agent or workgroup skills.
2. Map internal skill names to external skill names in an Interaction Processor table.
3. Set the routing context for Interaction Web Tools with the chatRoutingContext and callBackRoutingContext parameters.

> **Note:**
> You can optionally make the skill selection available to visitors on your website. This step requires you to write code that passes the visitor's choice to Interaction Web Tools. If you do not make a selection available, CIC routes interactions according to the context you set in the parameters.

4. (Optional) If not using the built-in "Language" and "Product" categories, customize the appropriate handlers to parse the visitor selections and route the interaction.

Uncomment the lines related to routing contexts:

```
var routingExample = new
WebServices.RoutingContexts("ExampleCategory","ExampleContext");
/* Must initialize with one category/context pair. Uncomment for routing
context.*/
routingExample.add("AnotherCategory","AnotherContext");
/* can add additional category/context pairs using add(). Uncomment for
routing context. */
```

Also uncomment the chatRoutingContext parameter:

```
"chatRoutingContext" : routingExample,
//uncomment for routing context.
```

The complete function might look like this example:

```
function setInteractionWebToolsParams(config, ui, WebServices) {
var routingExample = new
WebServices.RoutingContexts("ExampleCategory","ExampleContext");
routingExample.add("AnotherCategory","AnotherContext");
return {
"currentUriFragment" :
ui.ConfigConversions.convertICServerCountToCurrentUriFragment(config.ICServ
erCount),
"uriFragments" :
ui.ConfigConversions.convertICServerCountToUriFragments(config.ICServerCoun
t),
"pageMode" :
ui.ConfigConversions.convertInteractionTypesToPageMode(config.InteractionTy
pes),
"chatTarget" : config.ChatTarget,
"chatTargetType" : config.ChatTargetType,
"callbackTarget" : config.CallbackTarget,
"callbackTargetType" : config.CallbackTargetType,
"defaultLanguageCode" : config.DefaultLanguageCode,
// "customInfo" : null,
// "chatFollowupUrl" : null,
// "callbackAttributes" : null,
// "callbackRoutingContext" : null,
"chatRoutingContext" : routingExample,
"useHttps" : config.UseEncryption
};}
```

Writing a Custom Client

A web client can send several types of messages to `WebProcessorBridgeU.exe`.

Typically, the client sends the messages to a web server in the customer's DMZ (demilitarized zone). The web server in the DMZ then functions as a reverse proxy to the CIC server's `WebProcessorBridgeU.exe` process.

These requests can be in XML or JSON format. XML requests receive responses in XML; JSON requests receive responses in JSON. Though XML is available, Genesys recommends that you use JSON.

For more information about the message types, see the following:

HTTP Header Information	Poll for New Events	Common Response Elements
Message Type: Get Server Configuration	Retrieve a File Sent By Agent	Create a Callback
Query Queue	Send a Message	Query Callback Properties
Party Information	Update Typing Indicator	Query Callback Status
Start a Chat	Exit a Chat	Disconnect
Reconnect	Common Response Format	Modification

Unless otherwise specified, responses are in the **Common Response Format**.

> **Note:**
> All responses are formatted for clarity. Actual responses can contain more or fewer white space characters.

HTTP Header Information

When writing custom clients, use this HTTP header information:

- `Accept`: Use the `Accept` request-header field to specify media types that are acceptable for the response.
- `Accept-Charset`: Use the `Accept-Charset` request-header field to indicate which character sets are acceptable for the response.
- `Content-Range`: Send the `Content-Range` entity-header with a partial entity-body to specify where to apply the partial body in the full entity-body.
  Web Tools does not support Partial entity-body. Do not include the `Content-Range` entity-header in any requests.
- `Content-Type`: Use the `Content-Type` entity-header field to indicate the media type of the entity-body sent to the recipient.

If `Accept-Charset` is present, acceptable media types are:

- `application/json`
- `application/xml`
- `text/plain`
- `text/*`
- `/`

If `Accept` is not present but the `Content-Type` header is present and its value matches a listed type, then Web Tools uses that character set. Otherwise, Web Tools uses the default content type `application/json`. Web Tools uses UTF8 charset only. `Accept-Charset` should contain either `UTF8` or `*`.

Web Tools uses only `GET` and `POST` methods.

Message Type: Get Server Configuration

To identify a server's capabilities, use the `Get Server Configuration` message type.

> **Note:**
> In the following code, `en-us,en;q=0.7,fr;q=0.3` is just an example. In actual use, the value is whatever the browser sends to the web server as the value of the `Accept-Language` HTTP header.

Message	Code
JSON Request	GET /websvcs/serverConfiguration HTTP/1.1 **Accept:** application/json
JSON Response	(see code below)

```
[ { "serverConfiguration":
    { "capabilities":
      { "chat": [ "start",
                  "reconnect",
                  "poll",
                  "setTypingState",
                  "sendMessage",
                  "exit",
                  "supportAuthenticationTracker",
                  "supportAuthenticationAnonymous",
                  "transcript"
                ],
        "callback": [ "create",
                      "reconnect",
                      "status",
                      "disconnect",
                      "properties",
                      "modify",
                      "supportAuthenticationTracker",
                      "supportAuthenticationAnonymous"
                    ],
        "queueQuery": [ "supportAuthenticationTracker",
                        "supportAuthenticationAnonymous"
                      ],
        "common": [ "supportRegistrationTracker",
                    "partyInfo"
                  ]
      },
      "failoverURIs": [ ]
    }
  },
  { "browserAcceptLanguage": "ta,en-US;q=0.8,en;q=0.6"
  }
]
```

Query Queue

To find out the number of available agents in an ACD queue and the approximate wait time, send the `Query Queue` message

The `participant` block is required:
- For a non-anonymous but non-authenticated user, supply the name and specify `null` for the credentials.
- For an anonymous user, specify `Anonymous User` for the name and `null` for the credentials.
- For `queueType` always specify `Workgroup`.

Message	Code
JSON Request	```POST /websvcs/queue/query\n{\n "queueName": "Marketing",\n "queueType": "Workgroup",\n "participant":\n{\n "name": "Wally Webuser",\n "credentials": "s3cr3t"\n }\n}```
Successful JSON Response	```{\n "queue":\n {\n "agentsAvailable": 5,\n "estimatedWaitTime": 30,\n "custom" :\n {\n "attr1" : "value1",\n "attr2" : "value2"\n },\n "status":\n {\n "type": "success"\n }\n }\n}```
Failed JSON Response	```{\n "queue":\n {\n "status":\n {\n "type": "failure",\n "reason": "error.websvc.notACDQueue"\n }\n }\n}```

Party Information

To get the picture of the agent, send the `partyInfo` message. This message is available for both chats and callbacks.

The guid in the URL is the value of *participantID* of the web visitor, which was obtained earlier in the `Callback Create`, `Callback Reconnect`, or `Start Chat` response.

The guid in the POST data that is the value of the `participantID` key is the participantID of an agent whose name/picture the web user wants to see. This is obtained in the response to `Callback Status Query` or any chat response.

If no photo of the agent is available, the `photo` name/value pair is omitted.

Message	Code

JSON Request	<pre>POST /WebSvcs/partyInfo/7cbd650b-e8f5-45be-b1cc-7277671537b1
	
	{
	  "participantID": "1234abcd-0000-0000-aaaa-0123456789ab"
	}</pre>

Successful JSON Response	<pre>{
	  "partyInfo":
	  {
	    "name": "Alan Agent",
	    "photo": "/websvcs/abc/xyz.png",
	    "status":
	    {
	      "type": "success"
	    }
	  }
	}</pre>

Failed JSON Response	<pre>{
	  "partyInfo":
	  {
	    "status":
	    {
	      "type": "failure",
	      "reason": "error.websvc.something"
	    }
	  }
	}</pre>

Start a Chat

To begin a new chat, send the `Start a Chat` message. Currently, there is no concept of an external user joining an existing chat. You must get the server configuration before starting a chat.

The chat client gets the web user's name and passes it to the WebProcessor as part of the request.

The `participant` block is required:

- For a non-anonymous but non-authenticated user, supply the name and specify `null` for the credentials.
- For an anonymous user, specify `Anonymous User` for the name and `null` for the credentials.

The `attributes` block can have any number of children, but no grandchildren. The values of its children must be strings. You can optionally supply routing contexts. The contexts are sent to handlers, which look up skills in an Interaction Processor table and assign them to the interaction. You can use the `product` and `language` categories without modifying handlers. For more information, see the *Interaction Web Tools Technical Reference*.

If you use custom attributes, the server prefixes attribute names with `WebTools_`.

Interaction Web Tools supports the transcript feature if the server has transcript capability.

Message	Code
| **JSON Request**<br><br>**Note:**<br>`text/plain` is the only content type supported. | ```POST /WebSvcs/chat/start``` <br> ```json
{
  "supportedContentTypes": "text/plain",
  "participant":
  {
    "name": "Wally Webuser",
    "credentials": "s3cr3t"
  },
  "transcriptRequired": true,
  "emailAddress": "wally.webuser@nowhere.com",
  "target": "Marketing",
  "targettype": "Workgroup",
  "language": "en-us"
  "customInfo": "free-form text",
  "attributes" :
  {
    "some_name" : "some_string",
    "some_other_name" : "some_other_string"
    /* For security, the server prefixes
        attribute names with WebTools_. */
  },
  "routingContexts" :
  [
    { "context" : "ICM", "category" : "Product" },
    { "context" : "012789", "category" : "Accounts" }
  ]
}
``` |
| **Successful JSON Request** | ```json
{
  "chat":
  {
    "participantID": "6cbd650b-e8f5-45be-b1cc-7277671537b0",
    "chatID": "7cbd650b-e8f5-45be-b1cc-7277671537b9",
    "dateFormat": "M\/d\/yyyy",
    "timeFormat": "h:mm:ss tt",
    "status":
    {
      "type": "success"
    }
  }
}
``` |
| **Failed JSON Response** | ```json
{
  "chat":
  {
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.something"
    }
  }
}
``` |

Reconnect

To resume a chat on another CIC server after a switchover occurs, send the `Reconnect` message

The `participant` block is required:
- For an anonymous user, specify `Anonymous User` for the name and `null` for the credentials.
- For a non-anonymous but non-authenticated user, use the non-authenticated version of the Reconnect message.

Message	Code

JSON Request (non-authenticated user)	<pre>POST /websvcs/chat/reconnect
	{
	  "chatID": "6cbd650b-e8f5-45be-b1cc-7277671537b0"
	}</pre>

JSON Request (authenticated user)	<pre>POST /websvcs/chat/reconnect
	{
	  "chatID": "6cbd650b-e8f5-45be-b1cc-7277671537b0",
	  "participant":
	  {
	    "name": "John Doe",
	    "credentials": "s3cr3t"
	  }
	}</pre>

Successful JSON Response	<pre>{
	  "chat":
	  {
	    "participantID": " e9580ef2-1270-4182-a429-c3c0d957bf2a ",
	    "events":
	    [
	      {
	        "type": "participantStateChanged",
	        "participantID": " e9580ef2-1270-4182-a429-c3c0d957bf2a ",
	        "sequenceNumber": 0,
	        "state": "active",
	        "participantName": "John Doe"
	      },
	      {
	        "type": "participantStateChanged",
	        "participantID": "e9580ef2-1270-4182-a429-c3c0d957bf2a",
	        "sequenceNumber": 0,
	        "state": "active",
	        "participantName": "James"
	      }
	    ],
	    "status":
	    {
	      "type": "success"
	    }
	  }
	}</pre>

Failed JSON Response	<pre>{
	  "chat":
	  {
	    "status":
	    {
	      "type": "failure",
	      "reason": "error.websvc.something"
	    }
	  }
	}</pre>

Poll for New Events

Send the `Poll for New Events` message periodically to determine if anyone has joined/left the chat, if an agent has typed anything, if someone sent a file, and so forth.

> **Note:**
> If WebProcessorBridge doesn't hear from a session for longer than 2 minutes, the session is destroyed.

This message can only be sent when a chat is active. Specifically, after a successful response to a `Start a Chat` message, and before an `Exit a Chat` message is sent for that chat session. The message must include a guid in the URI. This guid is the value of `participantID` returned by the call to `Start a Chat`.

Message	Code
JSON Request	`GET /WebSvcs/chat/poll/c59f68d7-a1ff-416a-aa6e-09f978973a8a HTTP/1.0` `Accept: application/json`

Retrieve a File Sent By Agent

If the agent sends a file, it is placed in a *sandbox* on the server which runs `WebProcessorBridgeU.exe`. It can be retrieved by a typical HTTP request. The URI of the file is sent as part of the response to a poll, sent message, or the like.

Message	Code
Request	`GET /WebSvcs/chat/getFile/c59f68d7-a1ff-416a-aa6e-` `09f978973a8a/9b5d4caca0df42bbaab01e92f1126dbd/Sample0.txt HTTP/1.0`
Response	The contents of the requested file. If the file is not found, an HTTP 500 error is returned.

Send a Message

Use the `Send a Message` message when the remote user types a message for the agent.

This message can only be sent when a chat is active. Specifically, after a successful response to a `Start a Chat` message, and before an `Exit a Chat` message is sent for that chat session. It must include a guid in the URI. This guid is the value of `participantID` returned by the call to `Start a Chat`.

Message	Code

JSON Request	`POST /WebSvcs/chat/sendMessage/c59f68d7-a1ff-416a-aa6e-09f978973a8a HTTP/1.0`
	`Accept: application/json`
	
	`{`
	`  "message": "Can you please tell me my account balance?",`
	`  "contentType": "text/plain"`
	`}`

Update Typing Indicator

To indicate to the agent that the remote user is typing, or has stopped typing, use the `Update Typing Indicator` message.

This message can only be sent when a chat is active. Specifically, after a successful response to a `Start a Chat` message, and before an `Exit a Chat` message is sent for that chat session. It must include a guid in the URI. This guid is the value of `participantID` returned by the call to `Start a Chat`.

Message	Code

JSON Request	```POST /WebSvcs/chat/setTypingState/c59f68d7-a1ff-416a-aa6e-09f978973a8a HTTP/1.0```
	```Accept: application/json```
	
	```{ "typingIndicator": true }```

Exit a Chat

To stop the remote user's participation in a chat, use the `Exit a Chat` message.

This message can only be sent when a chat is active. Specifically, after a successful response to a `Start a Chat` message, and before an `Exit a Chat` message is sent for that chat session. It must include a guid in the URI. This guid is the value of `participantID` returned by the call to `Start a Chat`.

After this message is sent, no further messages are sent until or unless one of the following is sent:
- Get Server Configuration
- Start a Chat
- Create a Callback

Even though this message is an HTTP POST request, no data is posted.

Message	Code
JSON Request	```POST /WebSvcs/chat/exit/c59f68d7-a1ff-416a-aa6e-09f978973a8a HTTP/1.0``` ```Accept: application/json```

Common Response Format

The common response format is for responses to requests to:
- Start a Chat
- Poll for New Events
- Send a Message
- Update Typing Indicator
- Exit a Chat

This format controls how the system displays these types of information:
- How many milliseconds (always the same) to wait before polling again
- Events to indicate that someone joined or left the conversation
- Someone started or stopped typing
- Someone sent a message, file, or URL
- The participant ID of the web user joining the chat
- The status of the response
- Sequence numbers for overall events and for conversation events only:
  - sequenceNumber: the overall sequence number for events, including text and changes in participate state
- conversationSequenceNumber: the sequence number for only received text, URLs, and files

The listings illustrate all possible fields in this format. Not all fields are present in every response. Specifically:
- In responses to requests to start a chat, the WebProcessorBridge assigns a new participant ID to the web user (see the listings that follow, between the final event and the status). The WebProcessorBridge does not assign a new participant ID in any other

response types.

- In responses to chat exit requests, a subset of the format is used. `status`, `pollWaitSuggestion` and descendants are present, but `participantID` and its descendents are not present. `events` is ignored.
- Responses can include zero or more events.
- For events of type `participantStateChanged`:
  - `"state": "active"` indicates that the specified participant has joined the conversation.
  - `"state": "disconnected"` indicates that the specified participant has left the conversation.

Message	Code

Message	Code

JSON Response	`{`
	`  "chat":`
	`  {`
	`    "pollWaitSuggestion": 1000,`
	`    "events":`
	`    [`
	`      {`
	`        "type": "participantStateChanged",`
	`        "participantID": "f9580ef2-1270-4182-a429-c3c0d957bf2a",`
	`        "sequenceNumber": 0,`
	`        "state": "active",`
	`        "participantName": "John Doe"`
	`      },`
	
	`      {`
	`        "type": "text",`
	`        "participantID": "55b0e3cf-cecf-4f33-a6f7-cb1295fc9912",`
	`        "sequenceNumber": 1,`
	`        "conversationSequenceNumber": 0,`
	`        "contentType": "text\/plain",`
	`        "value": "Hello, how may I help you?",`
	`        "displayName": "Alan Agent"`
	`      },`
	
	`      {`
	`        "type": "typingIndicator",`
	`        "participantID": "55b0e3cf-cecf-4f33-a6f7-cb1295fc9912",`
	`        "sequenceNumber": 2,`
	`        "value": true`
	`      },`
	
	`      {`
	`        "type": "url",`
	`        "participantID": "55b0e3cf-cecf-4f33-a6f7-cb1295fc9912",`
	`        "sequenceNumber": 3,`
	`        "conversationSequenceNumber": 1,`
	`        "value": "http:\/\/www.inin.com\/",`
	`        "displayName": "Alan Agent"`
	`      },`
	
	`      {`
	`        "type": "file",`
	`        "participantID": "55b0e3cf-cecf-4f33-a6f7-cb1295fc9912",`
	`        "sequenceNumber": 4,`
	`        "conversationSequenceNumber": 2,`
	`        "value": "\/WebSvcs\/chat\/getFile\/e6edc474-19c5-42ea-9ea3-`
	`a4824e13cc7f\/73bd11cb1af54f7f9481cd7dba5c8e34\/Sample0.txt",`
	`        "displayName": "Alan Agent"`
	`      }`
	
	`    ],`
	`    "participantID": "f9580ef2-1270-4182-a429-c3c0d957bf2a",`
	`    "status":`
	`    {`
	`      "type": "success"`
	`    }`
	`  }`
	`}`

Message	Code
| JSON Response Indicating Failure | ```json
{
  "chat":
  {
    "pollWaitSuggestion": 1000,
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.content.invalid.missingData"
    }
  }
}
``` |

```json
  "chat":
  {
    "pollWaitSuggestion": 1000,
    "status":
```

Common Response Elements

Status

The `type` child of this element indicates whether the request succeeded or failed.

- If type is `failure`, then the `errorCode` element contains a hierarchical code specifying details of the failure. `pollWaitSuggestion` should be present. Other elements (events, pollWaitSuggestion) might be included, but it is unlikely.
- In a response to a request to exit a chat, `status` is the only relevant child of `chat`.

Events

This array can contain zero or more children. Each child must have a `type` element, which can be any of the following:

- participantStateChanged
- text
- file
- url
- typingIndicator

Each child must have a `participantID` element whose value is the guid of the participant who generated the event. Each child must have a `sequenceNumber` element which indicates this event's position in the list of all events for this chat.

- `type="participantStateChanged"`: This type of event appears once each time a participant changes state. Examples are becoming active and disconnecting. This type is specified as the value of the `state` element, and the corresponding examples of possible values are `active` and `disconnected`. This type of event can appear zero or more times in any response. The element can contain the display name of the participant who has changed state.
- `type="text"`: This type of event appears any time a chat participant has sent a textual message. It includes a `contentType` attribute to indicate to the server that the message being sent is plain text.
- `type="file"`: This type appears any time an agent has sent a file. The `"value"` attribute specifies the URI of the file to retrieve: it begins with `/WebSvcs` and the client can retrieve it by an HTTP GET request. If successful, the response is the contents of the file. If unsuccessful, the response is an HTTP 500 error.
- `type="url"`: This type appears any time a chat participant has sent a URL. The URL is specified by the `value` element.
- `type="typingIndicator"`: This type of event indicates that a participant has begun or stopped typing. If it does not appear for a specified participant, then:
  - If an event with `type="participantStateChanged"` element is present indicating that this participant has become active, then the user has not yet begun typing.
  - If an event with `type="text"` or `type="url"` exists for the same participant, then this participant is not typing (just finished typing a message)
  - If neither of these cases is true for a participant, then the user's typing status has not changed since the last response.
- Sequence numbers are without regard to event type. For example, if event 1 indicates that a participant has changed state, and a URL follows it, then the URL is 2, not a separate 1. Sequence numbers are scoped to the chat: for example, they do not reset to 1 with each new poll response.
- The GUI can use both sequence numbers and time stamps to display events in the proper order. It can use sequence numbers to determine if an event is missing.
- The `participantID` attribute specifies who generated the event. In the case of an event with `type="participantStateChanged"` indicating that a new participant has joined the chat, the participantID attribute allows the GUI to create an association between participant's guid and the participant's name.

pollWaitSuggestion

This response element is the number of milliseconds the WebProcessorBridge suggests the client wait before the next poll. The client is not required to follow this suggestion. The WebProcessorBridge uses the default interval of 2 seconds. You can, however, use the parameter in Interaction Administrator to specify a different interval. For more information, see the *Interaction Web Tools Technical Reference*.

> **Note:**
> You can use Interaction Administrator to customize the wait time suggestion. Navigate to **System Configuration**, then to **Web Services | Web Services Parameters**. Add a parameter named `MaxPollFrequency`.

Create a Callback

Create a Callback

To request that the CIC server create a callback interaction, send the `Create a Callback` message. You must get the server configuration before creating a callback: this message verifies that the server supports callbacks.

The "`participant`" block is required:

- For a non-anonymous but non-authenticated user, supply the name and specify `null` for the credentials.
- For an anonymous user, specify `Anonymous User` for the name and `null` for the credentials.

The "`attributes`" block can have any number of children, but no grandchildren. The values of its children must be strings.

The "`subject`" field has a maximum size of 2,000 characters.

The "`attributes`" block may have any number of children, but no grandchildren. The values of its children must be strings. You can optionally supply routing contexts. The contexts are sent to handlers, which look up skills in an Interaction Processor and assign them to the interaction. You can use the `product` and `language` categories without modifying handlers. For more information, see *Interaction Web Tools Technical Reference*.

Message	Code
| JSON Request | ```
POST /websvcs/callback/create
{
  "target": "Marketing",
  "targettype": "Workgroup",
  "subject": "baseball",
  "participant":
  {
    "name": "Wally Webuser",
    "credentials": "s3cr3t",
    "telephone": "555-5555"
  },
  "customInfo": "free-form text",
  "attributes" :
  {
    "some_name" : "some_string",
    "some_other_name" : "some_other_string"
},
  "routingContexts" :
  [
    { "context" : "ICM", "category" : "Product" },
    { "context" : "012789", "category" : "Accounts" }
  ],
  "language": "en-us"
}
``` |
| Successful JSON Response | ```
{
  "callback":
  {
    "participantID": "6cbd650b-e8f5-45be-b1cc-7277671537b0",
    "callbackID": "7cbd650b-e8f5-45be-b1cc-7277671537b9",
    "status":
    {
      "type": "success"
    }
  }
}
``` |
| Failed JSON Response | ```
{
  "callback":
  {
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.something"
    }
  }
}
``` |

Query Callback Properties

To retrieve a callback's subject and telephone number (the data that the user specifies), send the `Query Callback Properties` message. You can use the `Query Callback Properties` message to prepopulate the form for modifying a callback.

> **Note:**
> Compare this message to `Query Callback Status` which queries data ACD sets.

Message	Code
JSON Request	`GET /websvcs/callback/properties/6cbd650b-e8f5-45be-b1cc-7277671537b0`
Successful JSON Response	```{ "callback" : { "subject": "baseball", "telephone": "555-5555", "status" : { "type" : "success" } } }```
Failed JSON Response	```{ "callback": { "status": { "type": "failure", "reason": "error.websvc.something" } } }```

Successful JSON Response:
```
{
  "callback" :
  {
    "subject": "baseball",
    "telephone": "555-5555",
    "status" :
    {
      "type" : "success"
    }
  }
}
```

Failed JSON Response:
```
{
  "callback":
  {
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.something"
    }
  }
}
```

Query Callback Status

To retrieve status information on the callback's routing through CIC, send the `Query Status` message.

The guid in the URL is the value of `"participantID,"` which was obtained earlier.

> **Note:**
> Compare this message to `Query Callback Properties` which queries data that the user sets.

If the JSON response succeeds, the `"State"` value is `Active`.

When the scheduling callback feature is in place, `"State"` value is `Active` or `Inactive`.
- `Active` means callback is active on the ACD queue.
- `Inactive` means that the callback is waiting to be scheduled.

The `AcdStatus` block is omitted if the `"State"` is `Inactive`.

The value of `"queueWaitTime"` is the number of seconds the callback has been in the current queue. If the call is transferred, this value is reset to 0.

The value of `"estimatedCallbackTime"` is the number of seconds between now and the estimated time when an agent will call the creator back.

The value of `"longestWaitTime"` is in seconds, and designates the wait time of the interaction that has been in the queue longest.

If no agent has been assigned, the entire `"assignedAgent"` block is omitted.

Message	Code
| JSON Request | ```
GET /websvcs/callback/status/6cbd650b-e8f5-45be-b1cc-
7277671537b0
``` |
| Successful JSON Response (long-polling) | ```json
{
  "callback":
  {
    "state" : "Active",
    "assignedAgent":
    {
      "name": "Alan Agent",
      "participantID": "7cbd650b-e8f5-45be-b1cc-7277671537b1"
    },
    "acdStatus":
    {
      "queueWaitTime": 600,
      "interactionState": "ACD - Wait Agent",
      "estimatedCallbackTime": 3600,
      "queuePosition" : 1,
      "queueName" : "Sales",
      "longestWaitTime": 600,
      "callsWaitingCount": 4,
      "loggedInAgentsCount": 36,
      "availableAgentsCount" : 1
    },
    "status":
    {
      "type": "success"
    }
  }
}
``` |
| Failed JSON Response | ```json
{
  "callback":
  {
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.something"
    }
  }
}
``` |

Disconnect

To disconnect the callback, send the "Disconnect" message.

The guid in the URL is the value of "participantID," which was obtained earlier.

The body of this POST request is empty.

Message	Code
JSON Request	`POST /websvcs/callback/disconnect/6cbd650b-e8f5-45be-b1cc-7277671537b0`
| Successful JSON Response | ```json
{
 "callback":
  {
    "status":
    {
      "type": "success"
    }
  }
}
``` |
| Failed JSON Response | ```json
{
  "callback":
  {
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.something"
    }
  }
}
``` |

Modification

To modify the callback, send the "Modification" message.

Message	Code
| JSON Request | ```POST /websvcs/callback/modify/6cbd650b-e8f5-45be-b1cc-7277671537b0```<br><br>```json
{
  "subject": "baseball",
  "telephone": "555-5555",
  "customInfo": "free-form text",
  "attributes":
  {
    "some_name" : "some_string",
    "some_other_name" : "some_other_string"
  }
}
``` |
| Successful JSON Response | ```json
{
  "callback":
  {
    "status":
    {
      "type": "success"
    }
  }
}
``` |
| Failed JSON Response | ```json
{
  "callback":
  {
    "status":
    {
      "type": "failure",
      "reason": "error.websvc.something"
    }
  }
}
``` |

Notifications

The various components of Interaction Web Tools communicate with each other by means of Notifications. Some objects publish Notifications, and some objects subscribe to these Notifications. There are many types of Notifications, to convey things such as:

- A chat has started
- A participant has joined a chat
- A participant has sent a message
- The status of a callback

See the `INotification` class in `WebServices.js` for a full list of Notification types.

For more information about notifications, see the following:

- Subscribing to Notifications
- Unsubscribing from Notifications
- Custom Notification Types

Subscribing to Notifications

Consider the following class:

```
MyClass = Class.create(common.InterfaceImplementation /* …or a subclass thereof */,
{
    initialize : function($super) {
        $super();
        // …do things
    }
});
```

Making this class respond to Notifications is easy. Here, the class is notified when a message is received within a chat:

```
MyClass = Class.create(common.InterfaceImplementation /* …or a subclass thereof */,
{
    initialize : function($super) {
        $super();

        // Before registering as an observer, this object must declare
        // that it implements the appropriate observer interface.
        this.addImplementedInterface(webservices.Interfaces.IReceivedTextNotificationObserver, webservices);

        // Register as an observer, and begin receiving Notifications.
        webservices.NotificationRegistry.registerObserver(this,
webservices.Interfaces.IReceivedTextNotificationObserver);
    },

    // Fulfill the implementation contract of the
    // IReceivedTextNotificationObserver interface.
    processReceivedTextNotification : function(notification) {
        var text = notification.get_messageText();
        var sender = notification.get_participantId();
        var timestamp = notification.get_dateTime();
        // Do something with these…
    }
});
```

A single object can observe any number of Notification types, and a Notification type can be observed by any number of objects.

Unsubscribing from Notifications

If a Notification Observer wants to stop receiving Notifications, this can be accomplished as follows:

```
webservices.NotificationRegistry.unregisterObserver(this,
webservices.Interfaces.IReceivedTextNotificationObserver);
```

Custom Notification Types

In addition to the built-in Notification types, customers can create their own Notification types within Interaction Web Tools' Notification framework.

Declare a new Notification interface. The arguments are: its name, the names of its methods, the name of its superinterface, and the package containing the superinterface. This example Notification interface represents indication that a latitude and longitude have been mentioned in a chat.

```
 MyPackage.Interfaces.IGeoNotification = new common.Interface('MyPackage.Interfaces.IGeoNotification',
['get_latitude', 'get_longitude'], ['webservices.Interfaces.INotification'], webservices);
```

To implement this new interface, declare a Notification object:

```
MyPackage.GeoNotification = Class.create(webservices._Internal.NotificationBase,
{
    _className : "MyPackage.GeoNotification", // This is required!
    initialize($super, latitude, longitude)
    {
        $super();

        // This class implements the previously-created interface.
        this.addImplementedInterface(MyPackage.Interfaces.IGeoNotification);

        this.latitude = latitude;
        this.longitude = longitude;
    },

    // Implement the methods defined by the interface
    get_latitude()
    {
        return this.latitude;
    },

    get_longitude()
    {
        return this.longitude;
    }
}
```

Add this new Notification type to the NotificationRegistry so that it can be used. Pass the name of the interface, and a function that can create an instance of the class which implements that interface:

```
webservices.NotificationRegistry.registerNotificationType(MyPackage.Interfaces.IGeoNotification,
function(lat, lng) { return new MyPackage.GeoNotification(lat, lng); });
```

An instance of this Notification can be created and sent to observers as follows:

```
var geoNotification = webservices.NotificationFactory.createGeoNotification(39.8905251,-
86.2678351);

webservices.NotificationRegistry.process(geoNotification);
```

An object can be registered as an observer of this new Notification type in the way that was described in the previous section. For instance:

```
MapDisplayer = Class.create(common.InterfaceImplementation,
{
    initialize : function($super) {
        $super();

        // Before registering as an observer, this object must
        // implement the appropriate observer interface.
        this.addImplementedInterface(MyPackage.Interfaces.IGeoNotificationObserver, MyPackage);

        // Register as an observer, and begin receiving Notifications.
        webservices.NotificationRegistry.registerObserver(this,
MyPackage.Interfaces.IGeoNotificationObserver);
    },
```

```
    processGeoNotification : function(notification)
    {
        showMap(document.getElementById('map'), notification.get_latitude(), notification.get_longitude());
    },

    showMap : function(domElement, latitude, longitude)
    {
        // …
    }
});
```

Here is a class which observes ReceivedTextNotifications as in the previous section, and creates and sends GeoNotifications:

```
CoordsParser = Class.create(common.InterfaceImplementation /* …or a subclass thereof */,
{
    initialize : function($super) {
        $super();

        // Before registering as an observer, this object must
        // implement the appropriate observer interface.
        this.addImplementedInterface(webservices.Interfaces.IReceivedTextNotificationObserver, webservices);

        // Register as an observer, and begin receiving Notifications.
        webservices.NotificationRegistry.registerObserver(this,
webservices.Interfaces.IReceivedTextNotificationObserver);
    },

    // Fulfill the implementation contract of the
    // IReceivedTextNotificationObserver interface.
    processReceivedTextNotification : function(notification) {
        var text = notification.get_messageText();

        var coords = this.parseCoordsFromString(text);
        if (coords)
        {
            var geoNotification = webservices.NotificationFactory.createGeoNotification(coords.lat,
coords.long);
            webservices.NotificationRegistry.process(geoNotification);  // The Notification will be sent to
MapDisplayer
        }
    },

    parseCoordsFromString : function(string)
    {
        // …
    }
});
```

Sending and Receiving Webpages Using Handlers

You can use handlers to:

- Capture visitor-entered data from your website.

  You can create handlers that accept visitor-entered information, such as name, CIC password, and other types of information. The values entered are passed into handlers where they can be processed any way you like (for a form, hyperlink, and so on).

- Send information from handlers into custom webpages.

  Handlers processing the visitor-entered information can retrieve data from a database, CIC Directory Services, a mail server, or any other data source accessible to handlers. You can then use handlers to format that data and return it to the visitor in a custom webpage.

> **Example:**
> You could ask a visitor to enter the name or extension of a CIC agent. Your web server passes this information into a handler where you could generate a webpage showing that CIC agent's current availability and expected time of return.

For more information, see the following:

- [Capturing Visitor-entered Data from Your Website into Handlers](#)
- [Creating Handlers that Send Information to a Webpage](#)
- [Internet Tools, Initiators, and Handlers](#)

Capturing Visitor-entered Data from Your Website into Handlers

The following information describes the process by which information passes from a webpage to a handler.

Concepts introduced

- HTML event
- HTML Initiator
- WebHTMLService

You can start handlers with events generated by webpages. To work correctly, this process requires several correctly configured elements:

- A handler with an HTML event initiator
- A `doHTMLRequest` message sent from the web browser to WebHTMLService, part of CIC's HTTPPluginHost subsystem

  Use the same event name for both items. CIC watches for this event name and starts the correct handler when it receives a doHTMLRequest message with that event name.

- A URL rewrite that sends requests for use with handlers to either http://ICServer:8018/WebHTMLService/{R:1} or https://ICServer:8019/WebHTMLService/{R:1} for HTTP or HTTPS, respectively.

For example, `System_WebSearch.ihd` is a handler that is installed with CIC. It has an HTML event initiator with both the Registered Name and the HTML event fields set to WebSearch. It is started by sending a message to WebHTMLService that includes the following:

`doHTMLRequest?&event=WebSearch`

For example, a form could include this message as part of the submit action:

```
<input
  name="submit"
  type="s"
  value="OK"
  align=left
  onClick='ININ.Switchover.configureFormsForSwitchover("./WebServices/Server1/",



"./WebServices/Server2/", "doHTMLRequest?&event=WebSearch");'>
```

When the client sends a `doHTMLRequest` message, the web server passes the event name, form data, and any additional command-line arguments to the WebHTMLService on the CIC server. If CIC finds a handler with an HTML event initiator that uses

the WebSearch HTML event, the handler places any form data and command-line arguments into string variables, and processes the information.

The HTML initiator can start a unique handler for each unique HTML event your webpages generate.

The handler processes the information and sends a custom page back to the visitor over the same web connection. This entire transaction occurs through a proxy on your web server, so it works even if you are using a firewall to protect your network.

Example

This example shows how to look up a name and password submitted in an HTML form using the WebHTMLService.

The following example contains a form that allows a user to enter a user name and password. Behind the **Submit** button on the form is a call to WebHTMLService.

```
<html>
  <head>
    <title>Simple Example</title>
    <script type="text/javascript" src="./js/external/jquery.min.js"></script>
    <script type="text/javascript" src="./js/WebSearch/switchover.js"></script>
  </script>
  </head>
  <body>
    <form method="POST" >
      Enter your name and password:<br>
      <input name="Name" type="text"cols=50



size="50" align=left>
      <input name="Password"



type="text"cols=50 size="50" align=left>
      <input name="submit" type="s"



value="OK" align=left onClick='ININ.Switchover.configureFormsForSwitchover("./WebServices/Server1/",



"./WebServices/Server2/", "doHTMLRequest?&event=PasswordLookup&ARG1=French



");'>
    </form>
  </body>
</html>
```

When the visitor clicks the **Submit** button, WebHTMLService generates an event called PasswordLookup that contains the `Name` and `Password` string values. Notifier, a CIC subsystem that broadcasts events, receives the event, and broadcasts it to all of the other CIC subsystems. One of the subsystems notified is Interaction Processor. Interaction Processor starts any handlers with HTML initiators registered to start when the PasswordLookup event occurs.

A handler named `PasswordLookup.ihd` starts. The PasswordLookup initiator retrieves the values of `Name` and `Password` from the event and writes those values to variables that the handler can use.

As the example shows, you can pass two types of information: command-line data and form data.

For more information about capturing data, see the following:
- Command-Line Data
- Configuring the HTML Event Initiator to Receive Command-Line Data
- Form Data

Command-Line Data

Command-line data is information you want to pass into a handler, but that isn't entered by a user into a form. You can pass command-line data as arguments to the `doHTMLRequest` message sent to WebHTMLService. Any handler started by that event can extract the command-line data.

Syntax for Command-Line Data with WebHTMLService

The following example shows a hyperlink that calls WebHTMLService to generate the `EventTitle` event and pass the command-line parameter of `ARG1=French`.

```
<html>
  <head>
    <title>Simple Example</title>
    <script type="text/javascript" src="./js/external/jquery.min.js"></script>
    <script type="text/javascript" src="./js/WebSearch/switchover.js"></script>
    </script>
  </head>
  <body>
    <form method="POST">
      <input type="SUBMIT" value="Search" id=submit1 name=submit1
onClick='ININ.Switchover.configureFormsForSwitchover("./WebServices/Server1/", "./WebServices/Server2/",
"doHTMLRequest?event=EventTitle&ARG1=French");'>
    </form>
  </body>
</html>
```

Configuring the HTML Event Initiator to Receive Command-Line Data

When you call WebHTMLService, you specify an event name to start a handler. If you want these events to start a handler, configure that handler's HTML event initiator to start when that event occurs. The following graphic shows the **Event** page of the HTML event initiator:

When passing command-line arguments, you can ignore the **Field** or **Variable** sections of this page. Any values passed with the event go into a list-of-string variable that you name in the **CGI Command-Line Argument** field.

To configure the HTML event initiator with command-line arguments:

1. In Interaction Designer, open the HTML event initiator to the **Event** page.
2. Type a **Registered Name**.

   This name is stored in CIC along with any **Fields** and **Variables** you create.
3. Type the **HTML Event** name.

   For example, you might type `EventTitle`. Do not put quotes around this name.
4. Type a variable name for the **CGI Command-Line Arguments**.

   > **Note:**
   > This variable is a list-of-string value that holds all of the command-line data. The first command-line argument is assigned to list element 0, the second to list element 1, and so on.

5. Type a variable name for the **HTML Connection** value.

   The program uses this variable in the `Generate HTML` step that sends a webpage back to the person browsing your website.
6. Assign variables to the fields on the **Outputs** pages.

   Once you create these variables to hold the values passed in with the event, you can process them within your handler just like any other variables.

Form Data

Form data is information that a visitor types in an HTML form. When a website visitor enters data in an HTML form and submits it,

WebHTMLService packages the submitted form data in the event. An HTML event initiator can then extract the data.

Syntax for Form Data

The following is a form allowing a visitor to submit a name and password using the WebHTMLService. It generates the `PasswordLookup` event when a user clicks **Submit**:

```
<html>
  <head>
    <title>Simple Example</title>
    <script type="text/javascript" src="./js/external/jquery.min.js"></script>
    <script type="text/javascript" src="./js/WebSearch/switchover.js"></script>
    </script>
  </head>
  <body>
    <form method="POST" >
      Enter your name and password:<br>
      <input name="Name" type="text" cols=50 size="50" align=left>
      <input name="Password" type="text" cols=50 size="50" align=left>
      <input name="submit" type="submit" value="Submit" align=left
onClick='ININ.Switchover.configureFormsForSwitchover("./WebServices/Server1/", "./WebServices/Server2/",
"doHTMLRequest?&event=PasswordLookup");'>
    </form>
  </body>
</html>
```

As with command-line data in the previous HTML examples, specify the name of the event in the call to WebHTMLService. Any input values are bundled in the event, and the HTML event initiator can bind the data to a variable, where a handler can use it.

> **Note:**
> The variables that hold the visitor-entered values are list-of-string variables because some form elements, like check boxes or combo boxes (selection boxes), can pass more than one value into the event.

Configuring the HTML Event Initiator to Receive Form Data

When passing form data, bind the fields to the handler variables in the HTML event initiator. List the form elements from which you want to collect data, and assign a list-of-string variable to accept that data. Use these instructions as guidelines for configuring the HTML event initiator to accept form data.

To configure the HTML event initiator to accept form data:
1. Open the HTML event initiator to the **Event** page.
2. Type a **Registered Name**.
   This name contains the fields and variables you must create.
3. Type the HTML event name.
   For example, you might type `PasswordLookup`. Do not put quotes around this name.
4. Click **Add** to open the **HTML Tag Parameter** dialog box.
5. Type the name associated with an <input> element from your form.
   For example, you might type `Name` as your HTML Tag and then type a variable such as `NameList` to hold the value the visitor types in the form. The handler requires these list-of-string variables because some input types, such as check boxes and combo boxes, can pass more than one value. Repeat this process for each field you want to use in your handler.
6. Type a variable name in the **CGI Command-Line Arguments** field, although there are no values here when passing form data.
7. Type a variable name for the **HTML Connection** value.
   The program uses this variable in the `Generate HTML` step that sends a webpage back to the person browsing your website.
8. (Optional) Assign variables to the fields on the **Output** pages.

After you create these variables to hold the values passed in with the event, you can process them within your handler just like any other variables.

Creating Handlers that Send Information to a Webpage

The following information describes the process by which a handler sends a webpage back to a visitor browsing your website.

Concepts introduced

- Generate HTML tool
- Static webpage
- Custom webpage
- Substitution fields
- HTML template
- Binding
- Web connection

Just as you can pass information from a webpage into a handler, you can also pass information from a handler into a webpage. The Generate HTML tool can send a webpage to a person browsing your website over the web connection extracted by the HTML initiator.

A `Generate HTML` step can send two types of webpages back to visitors: static or custom.

- A static webpage is an HTML document that you have defined as a template. For static pages, the `Generate HTML` step sends a copy of the template back to the person browsing your website.
- A custom HTML document contains substitution fields to which you can bind variables from a handler. The `Generate HTML` step places values from the handler into the substitution fields defined in the template. Binding is the process by which values from handlers are associated with substitution fields in a template.

Example

This example shows how to return the status of an order in a custom webpage.

In the previous example, a person browsing your website submitted a name and password for handler processing. In this example, a handler sends a custom webpage back to that visitor containing the status of an order.

Using the name and password, `PasswordLookup` validates the visitor name and retrieves the status of the customer's order from a database. `PasswordLookup` then assigns the value to a string variable named `OrderStatus` and calls a `Generate HTML` step. This step performs several tasks:

1. Uses the web connection value retrieved by the HTML initiator and determines to whom to send the page.
2. Specifies the template to use to generate the webpage for the visitor.
3. Binds string variables from the handler to any substitution fields in the HTML template.

For our example, the `OrderStatus` is bound to a **Substitution Field** in the template. The **Template** and **Substitution Fields** appear in the following sample:

```html
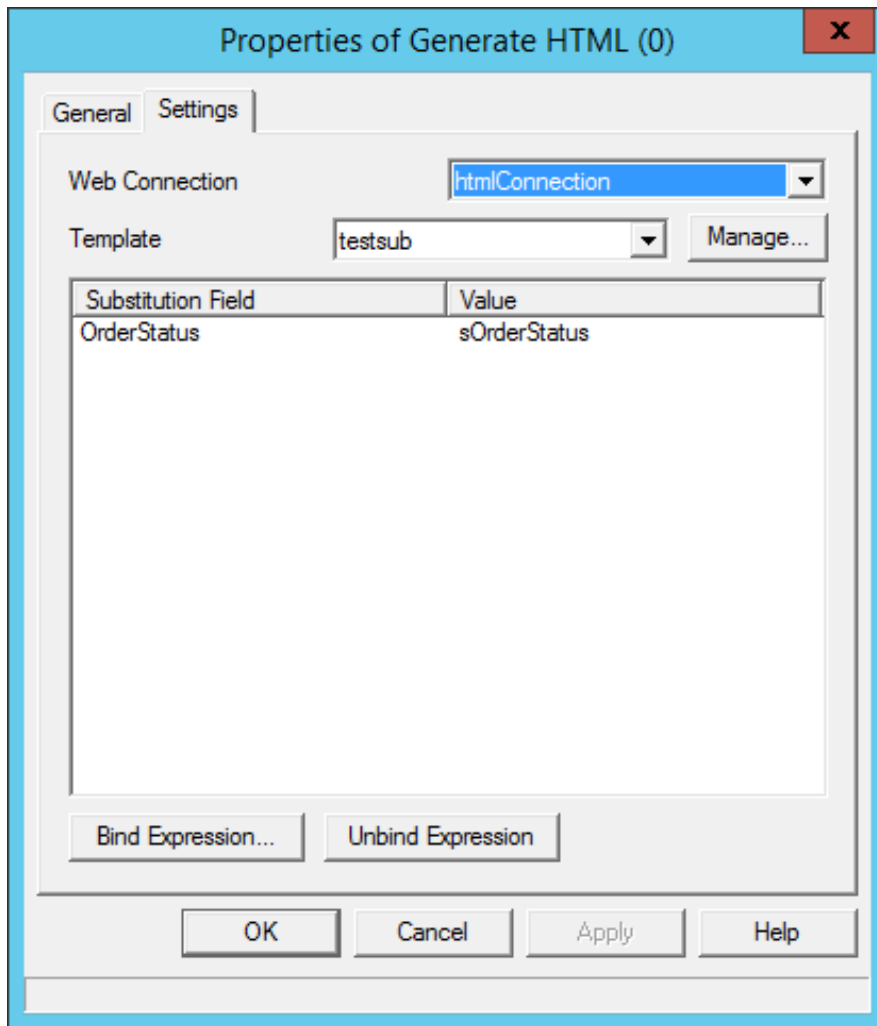<html>
  <head>
    <title>Simple Example</title>
  </head>
  <body>
    <h1>Order Status</h1>
    <div>
      Your order status: ##I3_OrderStatus##
    </div>
  </body>
</html>
```

When you specify this template in the `Generate HTML` step, `OrderStatus` appears in the **Substitution Field** list. You can bind the value of of each substitution field to a variable or other expression. That value is placed in the output that is created by the `Generate HTML` step.

As the `Generate HTML` step executes, the web server sends a page to the visitor, displaying the status of the order.

The key to creating handlers that send webpages back to visitors is the `Generate HTML` tool. This tool uses the web connection value gathered by the HTML event initiator to send a webpage back the visitor over the same connection.

> **Note:**
> Failure to send a webpage back to a visitor generates a webpage stating "Error Generating HTML" for that visitor.

For more information about sending data, see the following:

- Creating Templates
- Substitution Fields
- HTML Event Parameters
- Web HTML Handler Parameter

Creating Templates

Templates are HTML documents defined as templates by the `Generate HTML` tool. Any HTML document can be a template. Once you publish the handler containing the `Generate HTML` step, the program makes the text of the template part of the Java source code for that handler and writes it explicitly to the Java class file. No physical copy of the HTML template is required after that, unless you want to change the template.

Substitution Fields

Substitution fields enable you to pass string values from a handler into an HTML document. For example, use this feature to send visitor-specific information, such as account balances or the status of an order, to a person browsing your website.

When you create substitution fields in an HTML document, and then define that document as an HTML template, the substitution

fields appear in the `Generate HTML` step. There you can associate a string or list-of-string variable with a substitution field. The name for this process is binding.

Syntax for Substitution Fields

Substitution fields are easy to add to any HTML document. Add `##I3_SubstitutionFieldName##` to your document, and the `Generate HTML` tool parses the field when you define that document as a template.

```
<html>
  <head>
    <title>Simple Example</title>
  </head>
  <body>
    <h2>Today's Lucky Lotto Number is: ##I3_LottoNumberToday##</h2>
    <h2>Yesterday's Lucky Lotto Number was: ##I3_LottoNumberYesterday##</h2>
  </body>
</html>
```

Sample HTML substitution fields

This sample HTML code has two substitution fields, `##I3_LottoNumberToday##` and `##I3_LottoNumberYesterday##`. When the program saves code as an HTML document and selects it as a template, both **LottoNumberToday** and **LottoNumberYesterday** appear in the **Substitution Field** list in the `Generate HTML` tool.

You can also pass list-of-string values in to HTML lists. This table shows the different types of HTML lists that can contain substitution fields. Binding a list-of-string value to a substitution field in an HTML list produces a list item for every element in that list-of-string value.

List Type	Syntax

Ordered List	`<ol>`
	`   <li>##I3_SubList##</li>`
	`</ol>`

Bulleted (unordered) List	`<ul>`
	`   <li>##I3_SubList##</li>`
	`</ul>`

Definition List	`<dl>`
	`   <li>##I3_SubList##</li>`
	`<dl>`

Directory List	`<dir>`
	`   <li>##I3_SubList##</li>`
	`</dir>`

Interactive Menu List	`<menu>`
	`   <li>##I3_SubList##</li>`
	`</menu>`

Tables are another way to display list-of-string values to a person browsing your website. The following sample contains an example of a table that displays two substitution fields in table format.

```
<html>
  <head>
    <title>Simple Example</title>
  </head>
  <body>
    <h4>Withdrawals</h4>
```

```
   <table border="1">
     <tr>
       <td colstart="1">Date</td>
       <td colstart="2">Amount</td>
     </tr>
     <tr>
       <td colstart="1">##I3_WithdrawalDate##</td>
       <td colstart="2">##I3_WithdrawalAmount##</td>
     </tr>
   </table>
  </body>
</html>
```

> **Note:**
> If you place a substitution field in a list or table, the `Generate HTML` step requires that you bind a list-of-string variable to it from the handler. Likewise, if a substitution field is not contained within an HTML list or table element, the `Generate HTML` step requires that you bind to a single string value or variable.

Configuring the Generate HTML Tool (Binding Variables to Substitution Fields)

Once you have created an HTML document that you intend to use as a template, you must define that document as a template in the `Generate HTML` step. If you change your document, and want those changes reflected in the handler, perform this process again. Once you define an HTML document as a template, the substitution fields from that document appear in the `Generate HTML` step. You can bind variables from the handler to the substitution fields.

To configure the `Generate HTML` step:

1. Open the `Generate HTML` step to the **Settings** page.
2. In the **Web Connection** field, select the **HTML Connection** variable you created in the HTML event initiator that starts this handler.
3. Choose the template that you want this tool to send. If you haven't yet defined your HTML document as a template, click **Manage** to open the **HTML Template Registration** dialog. From there, you can name your template and select the HTML document you want to define as a template.
4. If you change this document after you define it as a template, repeat this process to redefine it and recognize any new substitution fields or HTML commands.
5. When you have chosen a template, all the substitution fields contained in the template appear in the **Substitution Field** list.
6. Bind each substitution field to a variable from the handler, or to an expression. Click **Bind Expression** to open the **Edit Expression** dialog box. In this dialog, select a variable from the handler to bind to the substitution field or build an expression. For more information about building expressions, see *Interaction Designer help*.

   > **Note:**
   > If you place a substitution field in a list or table, the `Generate HTML` step requires that you bind to it a list-of-string variable from the handler. Likewise, if a substitution field is not contained within an HTML list or table element, the `Generate HTML` step requires that you bind to a single string value or variable.

7. When you have bound all the substitution fields to variables from your handler or expressions, click **OK** to close the `Generate HTML` step, then save and publish your handler.

When you manage an HTML template in the **Generate HTML** step, it loads the HTML document into the Registry. You can view these forms under:

```
HKEY_LOCAL_MACHINE
  /SOFTWARE
    /Interactive Intelligence
      /EIC
        /Directory Services
          /Root
            /<SiteName>
              /Production
                /Configuration
                  /Web Forms
```

> **Warning!**
> Always use caution when viewing the Registry!

Once you complete these steps, you have configured the `Generate HTML` step to send a webpage back to a visitor browsing your website.

Any reference to a file and its location in an HTML document that begins with a `/` starts from the `WWWRoot` directory. Otherwise, CIC looks for this file in the current directory.

HTML Event Parameters

These parameters are automatically sent to the handler for its use. In addition, any custom parameters issued with the request in the form `&name=value` are sent to the handler as `name=value` parameters which the handler can parse and use.

Custom HTML Event	Description
AUTH_TYPE	HTTP request authorization type
CONTENT_LENGTH	HTTP request content type
CONTENT_TYPE	HTTP request content type
EVENT	Name of handler to initiate
HTTP_ACCEPT	HTTP Accept-Language header value
HTTP_COOKIE	Any cookies sent with the HTTP request
HTTP_USER_AGENT	HTTP header value for User-Agent
LOCALHOSTIP	IP address of web server
LOCALHOSTNAME	Name of web server
PATH_INFO	HTTP request path information
PATH_TRANSLATED	HTTP request path translated
QUERY_STRING	The request string separated from path to the program
REMOTE_ADDR	IP address of machine sending the request
REMOTE_HOST	Name of machine sending the request
REQUEST_METHOD	Set to Post
SCRIPT_NAME	Relative program path
SERVER_NAME	Name of web server
SERVER_PORT	Port used
SERVER_PROTOCOL	The HTTP (standard port 80) or HTTPS https (secure port 443)

Attributes

Optional Parameter	Type	Default Value	Description
CUSTOM_INFO	String	<none>	Used to pre-populate the registration form with information that customer entered in a previous visit. Strings of name value pairs are delimited by `|`.

Web HTML Handler Parameter

This configuration parameter is used by the Web HTML handler. Set it in **Interaction Administrator** > **System Configuration** > **Web Services** > **Web Services Parameters**.

Parameter	Description	Available
WebCGIRequestTimeout	This time out (in seconds) determines the maximum time a handler can take to reply to the CGI request. Default: 300	4.0 GA to present

Internet Tools, Initiators, and Handlers

For information about the tools, initiators, and handlers related to CIC Internet functionality, see the following:

- Web Interaction Tools
- Initiators
- Web-related Handlers
- Handler Attributes

For more detailed information, see the *Interaction Designer Help* at
https://help.genesys.com/cic/mergedProjects/wh_id/desktop/hid_introduction.htm.

Web Interaction Tools

Web Interaction tools are for building handlers that interact with people over the Internet. CIC can generate custom webpages and pop-up chat interfaces for a person browsing your website.

> **Note:**
> Web Interaction Tools are related to, but different from Interaction Web Tools. Tool is the term used by Interaction Designer for a component that can become a step in a handler. This section lists the tools in the **Web Interaction Tools** category in Interaction Designer.

Tool	Description
Alert Interaction	This Web Interaction tool notifies a station queue of a web interaction (chat, callback, etc.) that needs to be picked up. This tool alerts any station the interaction's recipient is logged into. If the interaction's recipient is a workgroup, user, or line queue, any recipient monitoring that queue from a CIC client is alerted. Alert Interaction times out if the interaction is not answered within a specific time period. Alert Interaction changes the state of an interaction to Alerting. Only interactions in a state of Offering, On Hold, or Voice Mail can be acted upon by the Alert Interaction tool. This tool also cancels any pending operations when transferring an ACD call to a user's queue.
Chat Goto URL	This tool sends a URL to participants of a web chat.
Chat Send File	This tool sends text to a file.
Conference Interaction	This tool creates a conference of two interactions, or adds an interaction to an existing conference
Consult Transfer	This tool transfers/merges the source interaction into the target interaction.
Create Interaction	This tool creates an interaction of specified type destined for the specified address (with the attributes provided),
Disconnect Interaction	This tool disconnects any web interaction.

Tool	Description
Hold Interaction	This tool places a connected web interaction on hold.
Mute Interaction	This tool mutes an interaction.
Park Interaction	Parks a web interaction (chats or callbacks) on a user queue you specify. The Park Interaction tool allows the handler to park an interaction again after the original park operation has timed out. This tool allows handlers to differentiate between parked interactions timing out and held interactions timing out. For example, when a parked chat timeout occurs, the timeout triggers the Held Call Timer initiator (just as it is for held calls). If an agent parks the call, the `ParkedFlag` output parameter in the Initiator is set to `true`. The handler could then give the caller the option to park the call again.
Pickup Interaction	This tool picks up a web interaction (chat or callback) that is on hold. It cannot pick up interactions that are not in a state of **On Hold**. This step does not pick up an interaction that a CIC client user has already picked up.
Receive Text	This tool receives text from a remote web interaction participant.
Receive Text Async	This tool receives text asynchronously from a remote web interaction participant. The "Complete Asynch Receive Text from an Interaction" initiator processes the text.
Record	Starts and stops recording of a web interaction, just as if a CIC client user had pressed the Record button. If this tool stops recording, the recording object is disconnected and the standard OnDisconnect handlers fire to process the recording and send it to the appropriate party.
Send Text	This tool sends text to a remote web interaction participant.
Send To Voicemail	This tool sends a chat or other web interaction into voice mail. **Note:** CIC does not support this feature for CIC client users with an Interaction Message Store (FBMC) mailbox.
Snooze Interaction	If one or more attempts to reach the callback requestor are not successful, an agent can decide to retry the callback request later. This puts the callback request into a *Snoozed* state. When the *Snooze* period ends, WebProcessor fires the Interaction Snooze Timed Out initiator which triggers a handler that starts ACD processing of this interaction.
Transfer Interaction	This tool transfers a web interaction from one queue to another. Valid queue types include User and Workgroup queues.

Initiators

Initiator	Description
Complete Async Receive Text from an Interaction	This initiator fires when it receives text or when the Receive Text Async tool times out. It processes text received asynchronously from a remote web interaction participant.
Interaction Transferred to Queue	When a web interaction is transferred to a non-system queue, it fires this initiator.
Messaging Request	When a web interaction is transferred to voice mail, it fires this initiator.
New Incoming Interaction	This initiator fires on all new incoming web interactions, such as chats or callbacks.
Interaction Snooze Timed Out	This initiator fires when the *Snooze* period for the interaction is over.

Web-related Handlers

Handler	Description
ACDAvailableInteraction	If the event is a chat or web callback, this subroutine calls ACDProcessEventInteraction.
ACDProcessEventInteraction	Assigns an ACD web interaction to an available agent and alerts that agent. If an agent is unavailable, it calls SystemACDInteractionHoldingProcess and places the chat in a queue for the next available agent.
System_IncomingInteraction	Receives a notification of a new incoming web interaction. These interactions include chats and web callbacks. Once received, the Blind Transfer tool step starts the System_InteractionOfferingNonSystemQueue handler which transfers the interaction to the appropriate queue.
System_InteractionOffering NonSystemQueue	Determines the type of queue (user or workgroup) for the web interaction and calls the appropriate subroutine.
System_InteractionVoicemail	When an agent sends the chat interaction to voice mail, the remote party sees text in the chat window that prompts them to type a message. When the remote party clicks **OK** to send the voice mail, this handler writes the typed message to the call log and emails the voice mail to the appropriate recipient. **Note:** CIC does not support this feature for CIC client users with an Interaction Message Store (FBMC) mailbox.
SystemACDInteractionHoldingProcess	Sends text to the remote chat client while the chat interaction is waiting for an agent to pick it up. The text includes an estimated hold time.
SystemInteractionDisconnect	This handler waits for web interactions to disconnect and writes out information to the call detail record for reporting.
SystemIVRUserQueueInteraction	This handler executes every time a web interaction is transferred to a local user. This process is similar to the process that routes telephone calls to a user.
SystemIVRWorkgroupQueueAlert	Called when the program puts an interaction on a workgroup queue, this handler determines the interaction type and alerts the workgroup.
SystemRecordInteraction	When called, queries the interaction type and starts recording.
SystemWorkgroupQueueInteraction	Determines the destination workgroup queue for a web interaction and starts ACD processing for the web interaction on that queue.

Handler Attributes

The following table lists the attributes used in Interaction Web Tools handlers:

Attribute Name	Description
Eic_AgentConnectedMsg	Message text sent to visitor when agent joins.
Eic_AgentDisconnectedMsg	Message text sent to visitor when agent disconnects.
Eic_CallbackPhone	The number of the person requesting a callback.
Eic_CallbackCompletion	This numeric value indicates whether a callback interaction completed successfully. It contains a value of 83 if the callback succeeded, or 70 if the callback failed.
Eic_CallbackCompletionDisplay	Contains a localized string description of the callback completion status. Its value is "S" if the callback succeeded, "F" if the callback failed, or "" if the completion has not been assigned.
Eic_CurrentURL	Current URL position of the web session.
Eic_CustomInfo	Custom information string passed from a webpage.
Eic_DisplayAttributes	List of attribute names displayed in the client.
Eic_DisplayName	The name displayed during chat interactions.
Eic_ExternalWebId	External Web Interaction ID.
Eic_OnHoldMsg	Message text sent when interaction placed on hold.
Eic_OwnerWebSession	ID of the web session.
Eic_Subject	Subject of the callback interaction.
Eic_VisitorConnectedMsg	Message text sent to agent when visitor joins.
Eic_VisitorDisconnectedMsg	Message text sent to agent when visitor disconnects.
Eic_WSEnableIdleTimeout	Set to 0 to disable timeout. Set to 1 to enable timeout. The timeout applies only to visitor (external user) activity.
Eic_WSIdleDisconnectMessage	Message text sent to the visitor after the chat is disconnected. The agent can also view this message.
Eic_WSIdleWarningMessage	Message sent to external user after the time defined in Eic_WSPartyIdleTime. The agent can also view this message.
Eic_WSPartyIdleGraceTime	The time (in seconds) the visitor can remain idle after the warning message is sent.
Eic_WSPartyIdleTime	The time (in seconds) the visitor can be idle before the Eic_WSIdleWarningMessage is sent.

Customizing the Agent Search Page

PureConnect provides an Agent Search page, which displays the list of agents who can receive chats, callbacks, or emails. When you implement the Agent Search page, you can designate which agents and images appear on it.

To customize the search page, modify the example search page included with Interaction Web Tools.

Interaction Web Tools uses the HTTPPluginHost CIC subsystem, not WebProcessorBridge, to communicate with handlers. HTTPPluginHost operates on a different port than WebProcessorBridge. Therefore, create another reverse proxy to use any handlers, including `System_WebSearch.ihd`, with Interaction Web Tools. For instructions about enabling handlers with Interaction Web Tools, see *Interaction Web Tools Technical Reference*.

Agent Search Page Example

Web authors can create a similar page by embedding a call to it within their HTML, as shown in this sample search page code:

```
<html>
  <head>
    <title>Simple Example</title>
    <script type="text/javascript" src="./js/external/jquery.min.js"></script>
    <script type="text/javascript" src="./js/WebSearch/switchover.js"></script>
    </script>
</head>

<body>
  <form method="POST" >
    First Name: <input size="15" name="FirstName" class="textinput"> <br>
    Last Name:  <input size="15" name="LastName" class="textinput" align="right"> <br>
    Department: <select name="Department">
               <option value="Marketing">Marketing</option>
               <option value="Sales"> Sales </option>
             </select>

    <input type="hidden" name="SearchFilterOut" value="">
    <input type="hidden" name="DefaultSearchWorkgroup" value="Marketing">
    <input type="hidden" name="PageTitle" value="Search Results">
```

```
      <input type="hidden" name="BannerImage" value="">
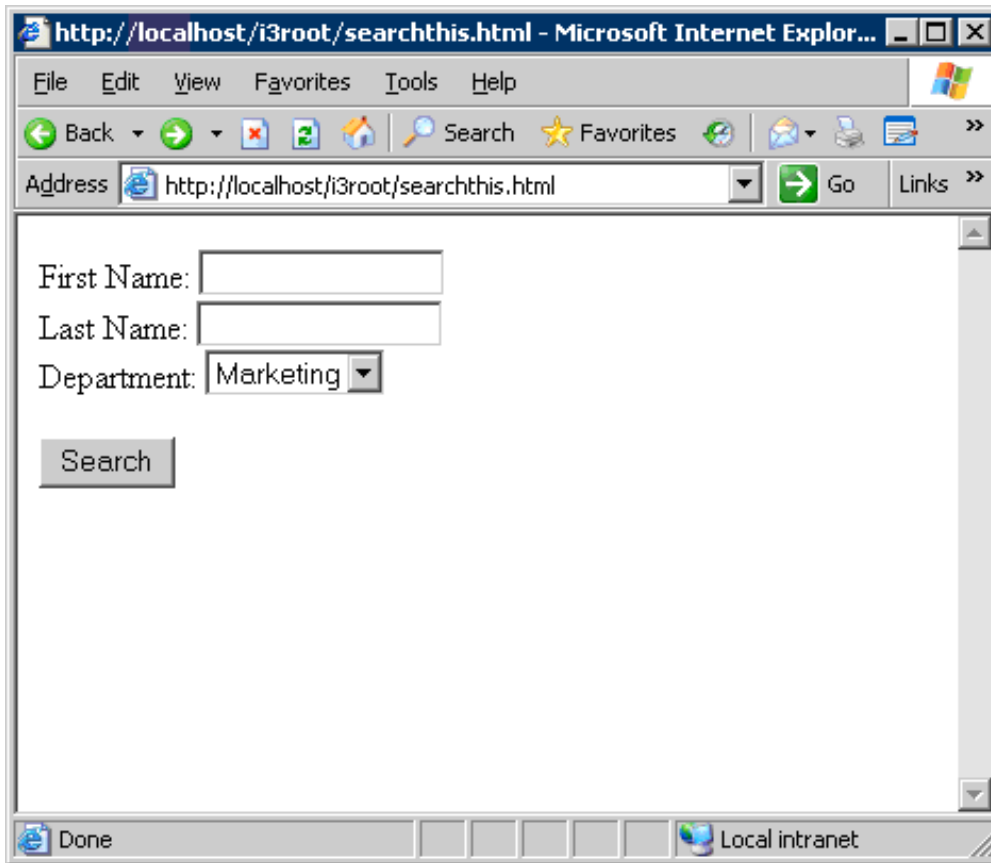      <input type="hidden" name="UserData" value="User Name&User@e-
mail.com&CompanyName&555.444.1234&Address1&Address2&Address3&City&State&ZipCode">
      <br>
      <br>
      <input type="SUBMIT" value="Search" id=submit1 name=submit1
onClick='ININ.Switchover.configureFormsForSwitchover("./WebServices/Server1/", "./WebServices/Server2/",
"doHTMLRequest?&event=WebSearch");'>
    </form>
  </body>
</html>
```

The following table defines the input arguments for the web search page.

Hidden Argument	Value
FirstName	The First Name value for the search
LastName	The Last Name value for the search.
Department	The Department value for the search.
PageTitle	Sets the title text on web search pages. The default page title is "Interact with Us!"
BannerImage	Specifies the banner image at the top of the page. The default is `Nav2Banner.gif`.

UserData	Passes data that you collect about the visitor, some of it critical to the performance of the web handlers. For example if a visitor asks for a callback request, the handlers need a telephone number. The web author is responsible to get this information from the visitor and put it into the following format with each value separated by an ampersand (`&`). The order is important. If a value is missing, hold its place with ampersands. For example
	
	```value="Name&e-mail&&phone&&&&&"```
	
	```Visitor data values:```
	```Name - The visitor's name. Required.```
	```E-mail address - The visitor's e-mail address. Required.```
	```Company Name - The visitor's Company.```
	```Phone Number - The visitor's phone number. Required.```
	``` (Verify the format of the number entered.)```
	```Address1 - The visitor's address.```
	```Address2 - More address information.```
	```Address3 - More address information.```
	```City - The visitor's city.```
	```State - The visitor's state.```
	```Zip Code - The visitor's Zip Code.```

SearchFilterOut	Specifies one or more CIC workgroups hidden from visitors when they specify a search. A visitor can search by first name, last name, or CIC workgroup. If this input argument is missing or left blank, then the visitor can choose to search from all CIC workgroups. If there are workgroups that you don't want displayed in the drop-down list box on the search option, list them in this argument, separated buy ampersands (`&`).
DefaultSearchWorkgroup	Specifies the default search workgroup in the drop-down list box of searchable workgroups. If missing or blank, no default workgroup appears in the drop-down list box.

The preceding code sample produces this search page:

Clicking **Search** produces the search results page, which the `System_WebSearch` handler generates.

You can customize the images by changing the images included with Interaction Web Tools. To customize other aspects of the search results page, customize the `System_WebSearch.ihd` handler.

Customizing the Images on the Agent Search Results Page

You can customize the appearance of the Agent Search page by changing the images that appear on it. In the folder where Interaction Web Tools is installed, locate the images for the Agent Search page in the `WebServices\img` subfolder.

> **Tip:**
> To simplify the customization process, keep the same file names.

Appendix A: Sample Session

This appendix gives a complete network transcript of a sample chat.

Request	Response	Description
```GET /websvcs/serverConfiguration HTTP/1.0```	```[
{
"serverConfiguration": {
"cfgVer": 1,
"capabilities": {
"chat": ["start" ,
"reconnect" , "poll" ,
"setTypingState" , "sendMessage"
, "exit" ,
"supportAuthenticationTracker" ,
"supportAuthenticationAnonymous"
, "transcript"],
"callback": ["create" ,
"reconnect" , "status" ,
"disconnect" , "properties" ,
"modify" ,
"supportAuthenticationTracker" ,
"supportAuthenticationAnonymous"
],
"queueQuery": [
"supportAuthenticationTracker" ,
"supportAuthenticationAnonymous"
],
"common": [
"supportRegistrationTracker" ,
"partyInfo" , "problemReport"]
},
"failoverURIs": [ ],
"problemReportRegEx":
"Chrome"
}
},
{ "browserAcceptLanguage":
"en-US,en;q=0.8" }
]```	Client requests server configuration
``POST /websvcs/chat/start HTTP/1.0

{
  "supportedContentTypes": "text/plain",
  "participant": {
    "name": "John Doe",
    "credentials": ""
  },
  "transcriptRequired": false,
  "emailAddress": "",
  "target": "Marketing",
  "targetType": "Workgroup",
  "clientToken": "deprecated",
  "language": "en-us"
}``` | ```{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "participantID": "94e8c3b0-
aeee-40e1-ba5a-a593c8047fc5",
    "dateFormat": "M\/d\/yyyy",
    "timeFormat": "h:mm:ss tt",
    "chatID": "73db8632-9ee8-
459e-a193-4d25684499d0",
    "status": { "type":
"success" }
  }
}``` | Client starts a chat. Server provides participant ID and chat ID The participant ID is used when sending future messages in the chat. However, if a switchover occurs, the chat ID will be needed to create a new participant ID on the other CIC server. |

Request	Response	Description

`GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`	`{`
	`  "chat": {`
	`    "pollWaitSuggestion": 2000,`
	`    "cfgVer": 1,`
	`    "events": [`
	`    {`
	`      "type": "participantStateChanged",`
	`      "participantID": "94e8c3b0-aeee-40e1-ba5a-a593c8047fc5",`
	`      "sequenceNumber": 0,`
	`      "state": "active",`
	`      "participantName": "John Doe"`
	`    }],`
	`    "status": {`
	`      "type": "success"`
	`    }`
	`  }`
	`}`
		Client polls. Server indicates that the web user has joined the chat.

Request	Response	Description

`GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`	`{`
	`  "chat": {`
	`    "pollWaitSuggestion": 2000,`
	`    "cfgVer": 1,`
	`    "events": [`
	`    {`
	`      "type": "text",`
	`      "participantID":`
	`"00000000-0000-0000-0000-`
	`000000000000",`
	`      "sequenceNumber": 1,`
	
	`"conversationSequenceNumber": 0,`
	`      "contentType":`
	`"text\/plain",`
	`      "value": "Welcome to`
	`CIC!",`
	`      "displayName": "IC",`
	`      "participantType":`
	`"System"`
	`    },`
	`    {`
	`      "type": "text",`
	`      "participantID":`
	`"00000000-0000-0000-0000-`
	`000000000000",`
	`      "sequenceNumber": 2,`
	
	`"conversationSequenceNumber": 0,`
	`      "contentType":`
	`"text\/plain",`
	`      "value": "Interaction`
	`transferred to Marketing.",`
	`      "displayName": "IC",`
	`      "participantType":`
	`"System"`
	`    },`
	`    {`
	`      "type": "text",`
	`      "participantID":`
	`"00000000-0000-0000-0000-`
	`000000000000",`
	`      "sequenceNumber": 3,`
	
	`"conversationSequenceNumber": 0,`
	`      "contentType":`
	`"text\/plain",`
	`      "value": "Interaction`
	`alerting Alan Agent.",`
	`      "displayName": "IC"`
	`    }],`
	`    "status": {`
	`      "type": "success"`
	`    }`
	`  }`
	`}`
		Client polls. Server sends some introductory text.

Request	Response	Description
| `GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [
    {
      "type":
"participantStateChanged",
      "participantID":
"3da146e0-0344-47a7-b9f5-
d88fafe98108",
      "sequenceNumber": 4,
      "state": "active",
      "participantName": "Alan
Agent",
      "participantType": "Agent"
    },
    {
      "type": "text",
      "participantID":
"00000000-0000-0000-0000-
000000000000",
      "sequenceNumber": 5,

"conversationSequenceNumber": 0,
      "contentType":
"text\/plain",
      "value": "CIC has joined
the conversation.",
      "displayName": "IC",
      "participantType":
"System"
    }],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server indicates that agent has joined the chat. |
| `GET /websvcs/chat/poll/94e8c3b0 HTTP/1.0` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [ ],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server indicates that nothing has happened since the previous poll.<br><br>Subsequent exchanges identical to this row are omitted from the remainder of this listing. |

Request	Response	Description

`GET /websvcs/chat/poll/94e8c3b0 HTTP/1.0`	<pre>{
	  "chat": {
	    "pollWaitSuggestion": 2000,
	    "cfgVer": 1,
	    "events": [
	    {
	      "type": "typingIndicator",
	      "participantID":
	"3da146e0-0344-47a7-b9f5-
	d88fafe98108",
	      "sequenceNumber": 6,
	      "value": true
	    }],
	    "status": {
	      "type": "success"
	    }
	  }
	}</pre>
		Client polls. Server indicates that agent is typing.

`GET /websvcs/chat/poll/94e8c3b0 HTTP/1.0`	<pre>{
	  "chat": {
	    "pollWaitSuggestion": 2000,
	    "cfgVer": 1,
	    "events": [
	    {
	      "type": "typingIndicator",
	      "participantID":
	"3da146e0-0344-47a7-b9f5-
	d88fafe98108",
	      "sequenceNumber": 7,
	      "value": false
	    },
	    {
	      "type": "text",
	      "participantID":
	"3da146e0-0344-47a7-b9f5-
	d88fafe98108",
	      "sequenceNumber": 8,
	
	"conversationSequenceNumber": 0,
	      "contentType":
	"text\/plain",
	      "value": "Hello, how may I
	help you?",
	      "displayName": "Alan
	Agent",
	      "participantType": "Agent"
	    }],
	    "status": {
	      "type": "success"
	    }
	  }
	}</pre>
		Client polls. Server sends agent's message and indicates that agent has stopped typing.

Request	Response	Description
| ```
POST
/websvcs/chat/setTypingState/94e8c3b0-
aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0

{
  "typingIndicator": true
}
``` | ```
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [ ],
    "status": {
      "type": "success"
    }
  }
}
``` | Client sets typing indicator to show that web user is typing. |
| ```
POST
/websvcs/chat/sendMessage/94e8c3b0-
aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0

{
  "message": "Can you please tell me my
account balance? My account number is
12345.\n",
  "contentType": "text/plain"
}
``` | ```
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "status": {
      "type": "success"
    },
    "events": [ ]
  }
}
``` | Client sends message from web user. |
| ```
GET /websvcs/chat/poll/94e8c3b0-aeee-
40e1-ba5a-a593c8047fc5 HTTP/1.0
``` | ```
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [
      {
        "type": "text",
        "participantID":
"94e8c3b0-aeee-40e1-ba5a-
a593c8047fc5",
        "sequenceNumber": 9,

"conversationSequenceNumber": 0,
        "contentType":
"text\/plain",
        "value": "Can you please
tell me my account balance?  My
account number is 12345.",
        "displayName": "John Doe",
        "participantType":
"WebUser",
      }],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server echoes message back to client. |

Request	Response	Description

`POST /websvcs/chat/setTypingState/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`
 	
`{`
`  "typingIndicator": false`
`}`
	`{`
	`  "chat": {`
	`    "pollWaitSuggestion": 2000,`
	`    "cfgVer": 1,`
	`    "events": [ ],`
	`    "status":`
	`    {`
	`      "type": "success"`
	`    }`
	`  }`
	`}`
		Client sets typing indicator to show that web user has finished typing.

`GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`	`{`
	`  "chat": {`
	`    "pollWaitSuggestion": 2000,`
	`    "cfgVer": 1,`
	`    "events": [`
	`    {`
	`      "type": "typingIndicator",`
	`      "participantID": "3da146e0-0344-47a7-b9f5-d88fafe98108",`
	`      "sequenceNumber": 10,`
	`      "value": true`
	`    }],`
	`    "status": {`
	`      "type": "success"`
	`    }`
	`  }`
	`}`
		Client polls. Server indicates that agent is typing.

Request	Response	Description
| `GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [
    {
      "type": "typingIndicator",
      "participantID":
"3da146e0-0344-47a7-b9f5-
d88fafe98108",
      "sequenceNumber": 11,
      "value": false
    },
    {
      "type": "text",
      "participantID":
"3da146e0-0344-47a7-b9f5-
d88fafe98108",
      "sequenceNumber": 12,

"conversationSequenceNumber": 0,
      "contentType":
"text\/plain",
      "value": "Your balance is
$678.90.  Is there anything else
I can help you with?",
      "displayName": "Alan
Agent",
      "participantType": "Agent"
    }],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server sends message from agent, and indicates that agent has stopped typing. |
| `POST /websvcs/chat/setTypingState/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`<br><br>`{`<br>`  "typingIndicator": true`<br>`}` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [ ],
    "status": {
      "type": "success"
    }
  }
}
``` | Client indicates that web user has begun typing. |
| `POST /websvcs/chat/sendMessage/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`<br><br>`{`<br>`  "message": "No thank you. Good bye!\n",`<br>`  "contentType": "text/plain"`<br>`}` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "status": {
      "type": "success"
    },
    "events": [ ]
  }
}
``` | Client sends message. |

Request	Response	Description
| `GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [
    {
      "type": "text",
      "participantID":
"94e8c3b0-aeee-40e1-ba5a-
a593c8047fc5",
      "sequenceNumber": 13,

"conversationSequenceNumber": 0,
      "contentType":
"text\/plain",
      "value": "No thank you.
Good bye!",
      "displayName": "John Doe",
      "participantType":
"WebUser"
    }],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server echoes message back to client. |
| `POST /websvcs/chat/setTypingState/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0`<br><br>```json
{
  "typingIndicator": false
}
``` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [ ],
    "status": {
      "type": "success"
    }
  }
}
``` | Client indicates that web user has finished typing. |
| `GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0` | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [
    {
      "type": "typingIndicator",
      "participantID":
"3da146e0-0344-47a7-b9f5-
d88fafe98108",
      "sequenceNumber": 14,
      "value": true
    }],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server indicates that agent has begun typing. |

Request	Response	Description
| GET /websvcs/chat/poll/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0 | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "events": [
    {
      "type": "typingIndicator",
      "participantID":
"3da146e0-0344-47a7-b9f5-
d88fafe98108",
      "sequenceNumber": 15,
      "value": false
    },
    {
      "type": "text",
      "participantID":
"3da146e0-0344-47a7-b9f5-
d88fafe98108",
      "sequenceNumber": 16,

"conversationSequenceNumber": 0,
      "contentType":
"text\/plain",
      "value": "Have a nice
day!",
      "displayName": "Alan
Agent",
      "participantType": "Agent"
    }],
    "status": {
      "type": "success"
    }
  }
}
``` | Client polls. Server sends message from agent, and indicates that agent has stopped typing. |
| POST /websvcs/chat/exit/94e8c3b0-aeee-40e1-ba5a-a593c8047fc5 HTTP/1.0 | ```json
{
  "chat": {
    "pollWaitSuggestion": 2000,
    "cfgVer": 1,
    "status": {
      "type": "success"
    },
    "events": [ ]
  }
}
``` | Client exits the chat. |

Appendix B: Customization Points

This appendix contains descriptions of the customization points.

Singleton Implementations

- LoginInfoSource - ui._Internal._DefaultLoginInfoSource
- MaximumFieldLengths - ui._Internal._DefaultMaximumFieldLengths
- RetryCounts - webservices._Internal._DefaultRetryCounts
- TabVisibility - ui._Internal._DefaultTabVisibility
- StatusFieldsDisplay - ui._Internal._DefaultStatusFieldsDisplay
- Linkifier - webservices._Internal._DefaultLinkifier
- SendOnEnter - webservices._Internal._DefaultSendOnEnter,
- ExtraCssClasses - ui._Internal._DefaultExtraCssClasses

Non-Singleton Implementation

- RegistrationFormPanel - ui._Internal._DefaultRegistrationFormPanel

LoginInfoSource

CustomLoginInfoSource class

In the default installation, a page is shown which allows the user to select between tabs for Chat, Callback, and Registration. To create a chat, the user must type their name (and optionally, a password) and press the **Start Chat** button. To create a callback, the user must type their name, telephone number, callback description, (and optionally a password), and press the **Start Callback** button.

This is because ui._Internal._DefaultLoginInfoSource is not able to get login information from any other source, so the default action is to show that page.

This subclass of _DefaultLoginInfoSource obtains login data from form submission data, and the login form is therefore not shown.

```
customizations.CustomLoginInfoSource = Class.create(ui._Internal._DefaultLoginInfoSource,
{
  initialize : function()
  {
    // Create an instance of the class that is defined just below this one.
    // This line could be placed elsewhere if desired, such as in chatOrCallback.html.
    var customLifecycleEventsObserverSingleton = new customizations.CustomLifecycleEventsObserver();
  },

  /**
   * Skip the login page, and begin a chat right away using a username
   * obtained from the form in this example's index.html
   */
  get_chatUsername : function()
  {
    // This line may be used to extract the value from the query string
    return this.get_queryStringValue("chatUsername");
  },

  /**
   * If get_chatUsername() returns non-null, this method may optionally be
   * used to return the password of that user.  If an anonymous chat is
   * desired, simply implement get_chatUsername() but allow get_chatPassword() to
   * return null.
   */
  get_chatPassword : function()
  {
    // This line may be used to extract the value from the query string (though that
    // is perhaps not a wise place for a password to be!)
    return this.get_queryStringValue("chatPassword");
```

```
  },

  /**
   * Skip the login page, and begin a callback right away using a username
   * obtained from the form in this example's index.html
   * Note that if get_chatUsername() also returns a non-null value, that will
   * take priority and a chat will be started, not a callback.
   */
  get_callbackUsername : function()
  {
    // This line may be used to extract the value from the query string
    return this.get_queryStringValue("callbackUsername");
  },

  /**
   * If get_callbackUsername() returns non-null, this method may optionally be
   * used to return the password of that user.  If an anonymous callback is
   * desired, simply implement get_callbackUsername() but allow get_callbackPassword() to
   * return null.
   */
  get_callbackPassword : function()
  {
    // This line may be used to extract the value from the query string (though that
    // is perhaps not a wise place for a password to be!)
    return this.get_queryStringValue("callbackPassword");
  },

  /**
   * If get_callbackUsername() returns non-null, this method shall return that user's
   * telephone number.
   */
  get_callbackTelephone : function()
  {
    // This line may be used to extract the value from the query string
    return this.get_queryStringValue("callbackTelephone");
  },

  /**
   * If get_callbackUsername() returns non-null, this method shall return the subject which
   * that user wishes to discuss.
   */
  get_callbackDescription : function()
  {
    // This line may be used to extract the value from the query string
    return this.get_queryStringValue("callbackDescription");
  },

  /**
   * The purpose of this example is to use usernames, passwords, etc. that were obtained from
   * an external source.  This is a helper method which uses the query string as that source.
   * Customers could easily replace this (or the calls to this) with code that gets the values
   * from other sources instead.
   *
   * @param key A key from a key+value pair in the query string
   */
  get_queryStringValue : function(key)
  {
    var value = common.Utilities.getQueryStringValue(key);

    if (null == value)
    {
      return null;
    }

    // Undo the URL encoding that was done when the form was submitted.
    // For instance, change "John+Doe" back to "John Doe"
    return decodeURIComponent(value.replace(/\+/g, ' '));
  }
});
```

MaximumFieldLengths

DefaultMaximumFieldLengths class

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.get_instance(webservices.CustomizableSingletonFactoryTypes.MaximumFieldLengths)

In the default installation, each text field within the 3 tabs will allow the user to enter up to the maximum number of characters that Tracker will support for that data type.

If it is desired to have a different maximum length for one or more fields, the following steps may be taken:

1. Create a subclass of this class. Override one or methods to return a different number.

   Note that it is not advisable to increase the returned values, as they are by default set to the maximum data length which Tracker can handle. Also note that this will have no effect on the pixel width of these fields - that can be changed by editing the .iwt-text-box selector of the CSS.

2. Change the line in customizations.MaximumFieldLengthsFactory that instantiates a new ui._Internal._DefaultMaximumFieldLengths. Make that line instead create an instance of the subclass from step 1.

```
ui._Internal._DefaultMaximumFieldLengths = Class.create(
{
  // Do not change these values (unless the Tracker DB Schema changes)
  TRACKER_USERNAME_MAXIMUM_LENGTH : 100,
  TRACKER_PASSWORD_MAXIMUM_LENGTH : 64,
  TRACKER_FIRST_NAME_MAXIMUM_LENGTH : 50,
  TRACKER_MIDDLE_NAME_MAXIMUM_LENGTH : 50,
  TRACKER_LAST_NAME_MAXIMUM_LENGTH : 50,
  TRACKER_NAME_MAXIMUM_LENGTH : 128,
  TRACKER_TELEPHONE_MAXIMUM_LENGTH : 255,
  TRACKER_SUBJECT_MAXIMUM_LENGTH : 2000,
  TRACKER_ADDRESS_MAXIMUM_LENGTH : 255,
  TRACKER_CITY_MAXIMUM_LENGTH : 50,
  TRACKER_STATE_MAXIMUM_LENGTH : 50,
  TRACKER_POSTAL_CODE_MAXIMUM_LENGTH : 20,
  TRACKER_COUNTRY_MAXIMUM_LENGTH : 50,
  TRACKER_EMAIL_MAXIMUM_LENGTH : 255,
  TRACKER_URL_MAXIMUM_LENGTH : 255,
  TRACKER_DEPARTMENT_MAXIMUM_LENGTH : 50,
  TRACKER_COMPANY_MAXIMUM_LENGTH : 100,
  TRACKER_JOB_TITLE_MAXIMUM_LENGTH : 100,
  TRACKER_REMARKS_MAXIMUM_LENGTH : 2000,

  /**
   * Constructor.  Does nothing.
   */
  initialize : function()
  {
  },

  /**
   * Override this method with one that returns a different number to alter the maximum
   * number of characters that a web user is allowed to type into a username field
   */
  get_usernameMaximumLength : function()
  {
    return this.TRACKER_USERNAME_MAXIMUM_LENGTH;
  },

  /**
   * Override this method with one that returns a different number to alter the maximum
   * number of characters that a web user is allowed to type into a password field
   */
  get_passwordMaximumLength : function()
  {
    return this.TRACKER_PASSWORD_MAXIMUM_LENGTH;
  },

  /**
   * Override this method with one that returns a different number to alter the maximum
   * number of characters that a web user is allowed to type into a first name field
```

```
 */
get_firstNameMaximumLength : function()
{
  return this.TRACKER_FIRST_NAME_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a middle name field
 */
get_middleNameMaximumLength : function()
{
  return this.TRACKER_MIDDLE_NAME_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a last name field
 */
get_lastNameMaximumLength : function()
{
  return this.TRACKER_LAST_NAME_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a name field (currently
 * used in two places: the name field of the web user when they choose "I don't have
 * an account", and the "Assistant Name" field).
 */
get_nameMaximumLength : function()
{
  return this.TRACKER_NAME_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a telephone (or fax, etc.) number field
 */
get_telephoneMaximumLength : function()
{
  return this.TRACKER_TELEPHONE_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a callback subject field
 */
get_subjectMaximumLength : function()
{
  return this.TRACKER_SUBJECT_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a street address field
 */
get_addressMaximumLength : function()
{
  return this.TRACKER_ADDRESS_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a city name field
 */
get_cityMaximumLength : function()
{
  return this.TRACKER_CITY_MAXIMUM_LENGTH;
},
```

```
/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a state (or province/territory) field
 */
get_stateMaximumLength : function()
{
  return this.TRACKER_STATE_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a postal code field
 */
get_postalCodeMaximumLength : function()
{
  return this.TRACKER_POSTAL_CODE_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a country field
 */
get_countryMaximumLength : function()
{
  return this.TRACKER_COUNTRY_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into an email field
 */
get_emailMaximumLength : function()
{
  return this.TRACKER_EMAIL_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a URL field
 */
get_urlMaximumLength : function()
{
  return this.TRACKER_URL_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a department name field
 */
get_departmentMaximumLength : function()
{
  return this.TRACKER_DEPARTMENT_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a company name field
 */
get_companyMaximumLength : function()
{
  return this.TRACKER_COMPANY_MAXIMUM_LENGTH;
},

/**
 * Override this method with one that returns a different number to alter the maximum
 * number of characters that a web user is allowed to type into a job title field
 */
get_jobTitleMaximumLength : function()
{
```

```
    return this.TRACKER_JOB_TITLE_MAXIMUM_LENGTH;
  },

  /**
   * Override this method with one that returns a different number to alter the maximum
   * number of characters that a web user is allowed to type into a remarks field
   */
  get_remarksMaximumLength : function()
  {
    return this.TRACKER_REMARKS_MAXIMUM_LENGTH;
  }

});
```

RetryCounts

DefaultRetryCounts class

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.get_instance(webservices.CustomizableSingletonFactoryTypes.RetryCounts)

If it is desired to change the number of times that failed AJAX requests of various types will be retried, do the following:

1. Create a subclass of webservices._Internal._DefaultRetryCounts. Override one or more methods to return a different number.
2. Change the line in RetryCountsFactory that instantiates a new webservices._Internal._DefaultRetryCounts. Make that line instead create an instance of the subclass from step 1.

> **Important!**
> Note that if certain HTTP response codes are received as a result of a request, the client will not retry the request, regardless of the value returned by the applicable method below.

```
webservices._Internal._DefaultRetryCounts = Class.create(
{
  /**
   * Constructor.  Does nothing.
   */
  initialize : function()
  {
  },

  /**
   * This method returns the number of times a failed poll operation
   * should be retried.
   *
   * Since a chat polls over and over again periodically, this defaults
   * to 0, because if one poll operation fails, another will happen in
   * a few seconds anyway.
   */
  get_pollRetries : function()
  {
    return 0;
  },

  /**
   * This method returns the number of times a failed request to exit
   * a chat should be retried.
   */
  get_exitRetries : function()
  {
    return 0;
  },

  /**
   * This method returns the number of times a failed request to start
   * a chat should be retried.
   */
  get_startChatRetries : function()
  {
    return 1;
```

```
    },

    /**
     * This method returns the number of times a failed request to reconnect
     * a chat should be retried.  (Applicable only if the WebProcessorBridge
     * lists reconnection as a capability, which is not the case at the
     * present time.)
     */
    get_reconnectRetries : function()
    {
      return 1;
    },

    /**
     * This method returns the number of times a failed request to register
     * a new account with Tracker should be retried.
     */
    get_trackerRegistrationRetries : function()
    {
      return 1;
    },

    /**
     * This method returns the number of times a failed request to get a file
     * sent by an agent should be retried.
     *
     * Not currently used, since clicking a link to get a file is not handled
     * via AJAX.
     */
    get_getFileRetries : function()
    {
      return 1;
    },

    /**
     * This method returns the number of times a failed request to create a
     * Callback should be retried.
     */
    get_createCallbackRetries : function()
    {
      return 1;
    },

    /**
     * This method returns the number of times a failed request to send a
     * message should be retried.
     */
    get_sendMessageRetries : function()
    {
      return 1;
    },

    /**
     * This method returns the number of times a failed request to set the
     * web user's typing state (to either true or false) should be retried.
     */
    get_setTypingStateRetries : function()
    {
      return 1;
    },

    /**
     * This method returns the number of times a failed request to obtain
     * server configuration should be retried, per server.
     *
     * Example: A switchover pair is configured.  For some reason, neither is responding.
     * A web user loads the chat client.  If this method returns 3 (the default),
     * the chat client will attempt to contact server #1, #2, #1, #2, #1, #2, #1,
     * #2, and then give up.  That reflects one try and three retries per server.
     *
```

```
 * Example 2: Switchover is not configured.  For some reason, the CIC server is
 * not responding.  A web user loads the chat client.  If this method returns
 * 3 (the default), the chat cilent will attempt to contact the CIC server
 * four times: one try and three retries.
 *
 * Note that this applies only to the attempt to obtain server configuration
 * before beginning a chat.
 *
 * This method does not apply to cases in which a chat is in-progress and one
 * of the other operations (poll, send message, etc.) fails for the specified
 * number of retries.  In that case, the chat client will attempt to get the
 * server configuration an indefinite number of times.  These attempts will
 * have a pause between them, the length of which is specified by the return
 * value of get_reconnectTimeoutMilliseconds().  In the case of a switchover pair,
 * it will try to connect to server #1, then server #2, then pause, and repeat this process
 * indefinitely.
 */
get_serverConfigurationRetries : function()
{
  return 3;
},

/**
 * This method returns the number of times a failed request to send a
 * problem report should be retried.
 */
get_problemReportRetries : function()
{
  return 0;
},

/**
 * If a chat that is in progress fails to connect to the
 * server (or, in the case of a switchover pair, fails to
 * connect to both of the servers), the chat will idle for
 * a period of time before attempting to contact the server(s)
 * again.  That period of time is determined by choosing a
 * random integer between the return value of this method and
 * its companion, get_reconnectTimeoutMillisecondsMaximum.
 *
 * If a non-random value is desired, modify RetryCountsFactory
 * to return a custom subclass of this class, and override both
 * methods to return the same value.
 *
 * Note: if a subclass overrides this method to return a different
 * value, it is recommended that the string associated with resource ID
 * "CouldNotConnectServerRetry" also be changed to reflect the new
 * timeout value.
 */
get_reconnectTimeoutMillisecondsMinimum : function()
{
  return 3000;
},

/**
 * If a chat that is in progress fails to connect to the
 * server (or, in the case of a switchover pair, fails to
 * connect to both of the servers), the chat will idle for
 * a period of time before attempting to contact the server(s)
 * choosing a random integer between the return value of this
 * method and its companion, * get_reconnectTimeoutMillisecondsMinimum.
 *
 * If a non-random value is desired, modify RetryCountsFactory
 * to return a custom subclass of this class, and override both
 * methods to return the same value.
 *
 * Note: if a subclass overrides this method to return a different
 * value, it is recommended that the string associated with resource ID
 * "CouldNotConnectServerRetry" also be changed to reflect the new
 * timeout value.
```

```
    */
  get_reconnectTimeoutMillisecondsMaximum : function()
  {
    return 5000;
  },

  /**
   * Sets how many milliseconds (thousandths of a second) to wait for a response
   * from an AJAX request before aborting the request.
   *
   * The default value is 12 seconds.
   */
  get_ajaxTimeoutMilliseconds : function()
  {
    return 12000;
  }
});
```

TabVisibility

DefaultTabVisibility class

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.get_instance(webservices.CustomizableSingletonFactoryTypes.TabVisibility)

By default:
1. The **Start Chat** tab is displayed if the Web Processor Bridge includes **start** and (**supportChatAuthenticationTracker** or **supportChatAuthenticationAnonymous**) in the list of chat capabilities (part of the server configuration response).
2. The **Start Callback** tab is displayed if **create** and (**supportCallbackAuthenticationTracker** or **supportCallbackAuthenticationAnonymous**) is included in the list of callback capabilities.
3. The **Register New Account** tab is displayed if **supportRegistrationTracker** is included in the list of common capabilities.

However, currently the Web Processor Bridge always includes all of the above. Therefore, this class (or a subclass thereof, depending on what customizations.TabVisibilityFactory returns) is queried to determine whether each tab should be shown or not.

To prevent certain tabs from displayed:
1. Create a subclass of this class which overrides one or more methods in this class.
2. Modify TabVisibilityFactory to return an instance of the new subclass instead of an instance of this class.

```
ui._Internal._DefaultTabVisibility = Class.create(
{
  /**
   * Constructor.  Does nothing.
   */
  initialize : function()
  {
  },

  /**
   * If a subclass overrides this return value to true, the "Start Chat" tab will
   * not be displayed.
   */
  hideStartChatTab : function()
  {
    return false;
  },

  /**
   * If a subclass overrides this return value to true, the "Start Callback" tab will
   * not be displayed.
   */
  hideStartCallbackTab : function()
  {
    return false;
  },

  /**
   * If a subclass overrides this return value to true, the "Register New Account" tab will
```

```
     * not be displayed, and the "Create an account" link on the other two tabs will also be hidden.
     */
    hideRegisterNewAccountTab : function()
    {
      return false;
    },

    /**
     * If this method returns false, the link to display a printable chat transcript
     * will be displayed.  If it returns true, the link will not be displayed.
     *
     * In the default implementation, false is returned.  However, subclasses may
     * override this method if the link is not (always) desired.
     *
     * If true is returned, the resource strings "ClosePageWarning" and "ExitPageWarning"
     * should be reworded, since they mention the ability to print a transcript.
     *
     * @return boolean indicating whether the printable chat history link should be hidden.
     */
    disablePrintableChatHistory : function()
    {
      return false;
    }
});
```

StatusFieldsDisplay

DefaultStatusFieldsDisplay class

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.get_instance(webservices.CustomizableSingletonFactoryTypes.StatusFieldsDisplay)

Controls whether the following fields are displayed in the Callback Status Panel. By default, all are displayed.
- Assigned Agent Name
- Interaction State
- Estimated Callback Time
- Queue Wait Time
- Queue Position
- Longest Wait Time
- Interactions Waiting Count
- Logged In Agents Count
- Available Agents Count
- Subject (entered by web user)
- Creation Time (time the callback request was submitted by web user)
- Web User's name (if anonymous) or username (if authenticated)
- Web user's telephone number

```
ui._Internal._DefaultStatusFieldsDisplay = Class.create(
{
  /**
   * Constructor.  Does nothing.
   */
  initialize : function()
  {
  },

  /**
   * This method returns whether the assigned agent's name should be displayed
   * in the callback status panel.
   *
   * @return Boolean
   */
  get_showAssignedAgentName : function()
  {
    return true;
  },
```

```
/**
 * This method returns whether the interaction state should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showInteractionState : function()
{
  return true;
},

/**
 * This method returns whether the assigned estimated callback time should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showEstimatedCallbackTime : function()
{
  return true;
},

/**
 * This method returns whether the queue wait time should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showQueueWaitTime : function()
{
  return true;
},

/**
 * This method returns whether the callback's position in the queue should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showQueuePosition : function()
{
  return true;
},

/**
 * This method returns whether the longest wait time of interactions in the queue should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showLongestWaitTime : function()
{
  return true;
},

/**
 * This method returns whether the number of interactions waiting on
 * the queue should be displayed in the callback status panel.
 *
 * @return Boolean
 */
get_showInteractionsWaitingCount : function()
{
  return true;
},

/**
 * This method returns whether the number of agents logged in should be displayed
 * in the callback status panel.
```

```
 *
 * @return Boolean
 */
get_showLoggedInAgentsCount : function()
{
  return true;
},

/**
 * This method returns whether the number of available agents should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showAvailableAgentsCount : function()
{
  return true;
},

/**
 * This method returns whether the callback's subject (as entered by the web user)
 * should be displayed in the callback status panel.
 *
 * @return Boolean
 */
get_showSubject : function()
{
  return true;
},

/**
 * This method returns whether the creation date/time of the callback should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showCreationDateTime : function()
{
  return true;
},

/**
 * This method returns whether the web user's name (if anonymous) or username (if authenticated)
 * should be displayed in the callback status panel.
 *
 * @return Boolean
 */
get_showName : function()
{
  return true;
},

/**
 * This method returns whether the web user's telephone number should be displayed
 * in the callback status panel.
 *
 * @return Boolean
 */
get_showTelephone : function()
{
  return true;
},

/**
 * Takes a prefix common to several resource keys, and a number of seconds, and
 * returns a localized string displaying that time duration.
 *
 * This is a customization point, to allow customers to tweak the number
 * of seconds before display.  This could be used to make the shortest displayed
 * time be 5 minutes, or to build in some over- or under-estimation, or to display
```

```
   * only increments of 5 minutes, etc. for instance.
   *
   * In this implementation, if seconds represents...
   * ...zero to 89 seconds, the returned value will be the resource
   * whose key is: resourcePrefix + "_Minute"
   * ...between 90 seconds and 45 minutes, the returned value will be the
   * rounded number of minutes substituted into the resource whose
   * key is: resourcePrefix + "_Minutes"
   * ...between 46 and 89 minutes, the returned value will be the resource
   * whose key is: resourcePrefix + "_Hour"
   * ...between 90 minutes and 20 hours, the returned value will be the rounded number
   * of hours substituted into the resource whose key is: resourcePrefix + "_Hours"
   * ...at least 20 hours but less than 36 hours, the returned value will be the
   * resource whose key is: resourcePrefix + "_Day"
   * ...at least 36 hours, the returned value will be the rounded number of days substituted
   * into the resource whose key is: resourcePrefix + "_Hours"
   *
   * In a later SU, this method will be changed to correctly handle the special rules for writing
   * plural numbers in languages such as Russian and Polish.
   *
   * @param resourcePrefix - A prefix common to several keys in the resource file. This method may append
"_Minute", "_Minutes", "_Hour", "_Hours".
   * @param seconds - integer number of seconds
   * @return Localized string
   */
  formatTimeDuration : function(resourcePrefix, seconds)
  {
    var timeDuration = new webservices.TimeDuration(seconds);
    var resourceSuffix = "";

    if (timeDuration.getTotalSeconds() <= 89)
    {
      return localization[resourcePrefix + "_Minute"];
    }
    else if (timeDuration.getTotalMinutes() <= 45)
    {
      var nMinutesToDisplay = timeDuration.getRoundedMinutes();
      return localization[resourcePrefix + "_Minutes"].replace('%0', nMinutesToDisplay);
    }
    else if (timeDuration.getTotalMinutes() <= 89)
    {
      return localization[resourcePrefix + "_Hour"];
    }
    else if (timeDuration.getTotalHours() <= 20)
    {
      var nHoursToDisplay = timeDuration.getRoundedHours();
      return localization[resourcePrefix + "_Hours"].replace('%0', nHoursToDisplay);
    }
    else if (timeDuration.getTotalHours() <= 36)
    {
      return localization[resourcePrefix + "_Day"];
    }
    else
    {
      nDaysToDisplay = timeDuration.getRoundedDays();
      return localization[resourcePrefix + "_Days"].replace('%0', nDaysToDisplay);
    }
  }
});
```

Linkifier

DefaultLinkifier class

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.get_instance(webservices.CustomizableFactoryTypes.Linkifier)

Scans text for URIs (http, https, ftp, file, mailto) and inserts the appropriate HTML to make them become links

Note that this will NOT linkify "www.somewhere.com" - the scheme part (for instance, "http://") is necessary for linkification. This may be corrected in a future SU.

```
webservices._Internal._DefaultLinkifier = Class.create(
{
  _linkOpeningTagPrefix : '<a href="',
  _linkOpeningTagSuffix : '" target="_blank" class="iwc-message-link">',
  _linkClosingTag : '</a>',
  _hideSchemeFromUser : true,

  // The following regular expression will match URLs specified with a
  // scheme (e.g. http://www.genesys.com or http://www.genesys.com/directory/file?key=value
  // or ftp://www.genesys.com or mailto:support@genesys.com).
  // It will also match URLs with no scheme specified (e.g. www.genesys.com), IF they
  // are of the form "www dot something"
  //                                (         scheme            )(URL) |(   www...    )
//IC-123488: regex changed to terminate on (), in addition to whitespace, and to terminate on a '.' not
followed by a matching char.
  _urlMatchingRegularExpression: /(?:(?:((?:(?:https?|ftp):\/\/)|(?:mailto:))((\.?[^\s\(\)\.,])+))|(www\.
[^\.\s](\.?[^\s\(\)\.,])+))/ig,

/**
 * Constructor
 */
  initialize : function()
  {
  },

  // public methods

  /**
 * Scans text for URIs (http, https, ftp, file, mailto) and inserts the appropriate HTML to make them become
links
 *
 * @param text The text to "linkify"
 * @return The text with URLs converted to links
 */
  linkifyText : function(text)
  {
    return text.replace(this.getUrlMatchingRegularExpression(), this._onMatch.bind(this));
  },

  /**
   * Creates a hyperlink, using this class' defined tags.
   *
   * Example:  Depending on the values of this._linkOpeningTagPrefix, etc.,
   * createLink("http://www.genesys.com", "Genesys") may return
   * <a href="http://www.genesys.com">Genesys</a>
   *
   * @param url The URL that the link points to
   * @param text The text for the user to click on.  If not specified, will default to the value of url.
   * @return string containing an HTML "a" tag (opening tag, text for the user to click on, and closing tag)
   */
  createLink : function(url, text)
  {
    return this.getLinkOpeningTagPrefix() + url + this.getLinkOpeningTagSuffix() + (text || url) +
this.getLinkClosingTag();
  },

  /**
   * The Linkifier inserts HTML "a" tags into the text.
   * This method returns the portion of the "a" tag that comes before the URL.
   *
   * @return string
   */
  getLinkOpeningTagPrefix : function()
  {
    return this._linkOpeningTagPrefix;
  },
```

```
  /**
   * The Linkifier inserts HTML "a" tags into the text.
   * This method returns the portion of the "a" tag that comes after the URL.
   *
   * @return string
   */
  getLinkOpeningTagSuffix : function()
  {
    return this._linkOpeningTagSuffix;
  },

  /**
   * The Linkifier inserts HTML "a" tags into the text.
   * This method returns the "/a" tag
   *
   * @return string, by default "</a>"
   */
  getLinkClosingTag : function()
  {
    return this._linkClosingTag;
  },

  /**
   * Returns the regular expression used to identify URLs in text.
   *
   * @return regular expression
   */
  getUrlMatchingRegularExpression : function()
  {
    return this._urlMatchingRegularExpression;
  },

  /**
   * Returns whether to hide "http://" and "mailto:" from
   * the user when displaying clickable URLs
   *
   * @return Boolean
   */
  getHideSchemeFromUser : function()
  {
    return this._hideSchemeFromUser;
  },

  // Private methods

  /**
   * If we've found a URL that was specified with a scheme, e.g. "https://www.genesys.com", then:
   * fullURL = URL including scheme, e.g. "https://www.genesys.com"
   * scheme = scheme, e.g. "https://"
   * afterScheme = URL without scheme, e.g. "www.genesys.com"
   *
   * But if we've found a URL that was specified without a scheme, e.g. "www.genesys.com", then:
   * fullURL = what was found, e.g. "www.genesys.com"
   * scheme = null
   * afterScheme = ALSO NULL!
   */
  _onMatch : function(fullURL, scheme, afterScheme)
  {
    if (!scheme)
    {
      // Found a URL that was specified without a scheme, e.g. "www.genesys.com"
      // Fix up the values so that they are what they'd be if scheme had been provided.
      scheme = "http://";
      afterScheme = fullURL;
      fullURL = scheme + afterScheme;
    }

    // Only allow hiding of scheme from user if it is http or mailto.  Still show it if it is https or ftp.
    if (this.getHideSchemeFromUser() && ("http://" == scheme.toLowerCase() || "mailto:" ==
scheme.toLowerCase()))
```

```
      {
        return this.createLink(fullURL, afterScheme);
      }
      else
      {
        return this.createLink(fullURL);
      }
    }
});
```

ExtraCssClasses

DefaultExtraCssClasses class

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.get_instance(webservices.CustomizableSingletonFactoryTypes.ExtraCSSClasses)

Interaction Web Tools uses Bootstrap (http://www.getbootstrap.com) for layout. However, some customers may wish to not use Bootstrap, or may wish to change the specific ways in which Bootstrap is used. Using IWT's customization framework to replace this class with a different one is the way to make that change.

```
ui._Internal._DefaultExtraCssClasses = Class.create(
{
  // Set up some default grid column widths for our forms
  _default_sm_label : 3,
  _default_sm_optional_label : 2,
  _default_md_label : 3,
  _default_md_optional_label : 2,
  _default_lg_label : 2,
  _default_lg_optional_label : 1,

  /**
   * Constructor.  Sets up a mapping of IWT's CSS classes and IDs to Bootstrap's CSS classes.
   */
  initialize : function()
  {
    this._default_sm_textbox = 12 - (this._default_sm_label);
    this._default_sm_optional_textbox = 12 - (this._default_sm_label + this._default_sm_optional_label);
    this._default_md_textbox = 12 - (this._default_md_label);
    this._default_md_optional_textbox = 12 - (this._default_md_label + this._default_md_optional_label);
    this._default_lg_textbox = 12 - (this._default_lg_label);
    this._default_lg_optional_textbox = 12 - (this._default_lg_label + this._default_lg_optional_label);

    this._map =
    {
      '.iwt-form-tabs': ['nav', 'nav-tabs'],
      '.iwt-tab-content': ['tab-content'],
      '#iwt-chat-form-panel': ['tab-pane'],
      '#iwt-callback-container-panel': ['tab-pane'],
      '#iwt-register-form-panel': ['tab-pane'],
      '.iwt-form': ['form-horizontal'],
      '.iwt-form-field-div': ['form-group'],
      '.iwt-radio': ['radio'],
      '.iwt-checkbox': ['checkbox'],
      '.iwt-textbox': ['form-control'],
      '.iwt-subject-textarea': ['form-control'],
      '.iwt-form-submit-button': ['btn', 'btn-lg', 'btn-primary'],

      '.iwt-form-label': ['col-sm-'+this._default_sm_label, 'col-md-'+this._default_md_label, 'col-lg-
'+this._default_lg_label, 'control-label'],
      '.iwt-optional-label': ['col-sm-'+this._default_sm_optional_label, 'col-sm-push-
'+this._default_sm_optional_textbox, 'col-md-'+this._default_md_optional_label, 'col-md-push-
'+this._default_md_optional_textbox, 'col-lg-'+this._default_lg_optional_label, 'col-lg-push-
'+this._default_lg_optional_textbox],
      '.iwt-textbox-container': ['col-sm-'+this._default_sm_textbox, 'col-md-'+this._default_md_textbox, 'col-
lg-'+this._default_lg_textbox],
      '.iwt-textbox-container-optional': ['col-sm-'+this._default_sm_optional_textbox, 'col-sm-pull-
'+this._default_sm_optional_label, 'col-md-'+this._default_md_optional_textbox, 'col-md-pull-
'+this._default_md_optional_label, 'col-lg-'+this._default_lg_optional_textbox, 'col-lg-pull-
```

```
'+this._default_lg_optional_label],
      '.iwt-form-field-no-label': ['col-sm-'+this._default_sm_textbox, 'col-sm-offset-
'+this._default_sm_label, 'col-md-'+this._default_md_textbox, 'col-md-offset-'+this._default_md_label, 'col-
lg-'+this._default_lg_textbox, 'col-lg-offset-'+this._default_lg_label],
      '.iwt-form-field-error': ['col-sm-'+this._default_sm_textbox, 'col-sm-offset-'+this._default_sm_label,
'col-md-'+this._default_md_textbox, 'col-md-offset-'+this._default_md_label, 'col-lg-
'+this._default_lg_textbox, 'col-lg-offset-'+this._default_lg_label],
      '.iwt-form-button-div': ['col-sm-'+this._default_sm_textbox, 'col-sm-offset-'+this._default_sm_label,
'col-md-'+this._default_md_textbox, 'col-md-offset-'+this._default_md_label, 'col-lg-
'+this._default_lg_textbox, 'col-lg-offset-'+this._default_lg_label],

      '.iwt-callback-status-panel': ['iwt-contains-floating-child'],
      '.iwt-callback-notices-container': ['row'],
      '.iwt-callback-subject-and-status-indicator-container': ['row'],
      '.iwt-callback-status-and-avatar-container': ['row'],
      '.iwt-callback-disconnect-button-panel-container': ['row'],
      '.iwt-callback-failure-panel-container': ['row'],
      '.iwt-callback-status-key': ['col-xs-6 control-label'],
      '.iwt-callback-status-value': ['col-xs-6'],
      '.iwt-callback-status-field': ['form-control-static'],
      '.iwt-callback-participant-avatar-div': ['col-sm-3'],
      '.iwt-callback-failure-panel': ['col-sm-12'],
      '.iwt-callback-status-fields-container': ['col-sm-9'],
      '.iwt-callback-status-form': ['form-horizontal'],
      '.iwt-callback-disconnect-button': ['btn', 'btn-danger', 'btn-lg'],
      '.iwt-callback-creation-success-panel': ['col-xs-12'],
      '.iwt-callback-disconnected-panel': ['col-xs-12'],
      '.iwt-callback-status-subject-div': ['col-sm-9'],
      '.iwt-callback-status-indicator-container': ['col-sm-3'],
      '.iwt-callback-disconnect-button-panel': ['col-sm-3', 'col-sm-offset-3'],
      '.iwt-callback-status-failure-container': ['col-sm-12'],

      '.iwt-chat-participants-panel': ['row'],
      '.iwt-chat-printable-history-link': ['pull-right'],
      '.iwt-print-div': ['col-lg-3', 'col-lg-push-9', 'col-sm-4', 'col-sm-push-8'],
      '.iwt-participants-panel-list-container': ['col-lg-9', 'col-lg-pull-3', 'col-sm-8', 'col-sm-pull-4'],
      '.iwt-chat-participant-popover-content': ['iwt-contains-floating-child'],
      '.iwt-message-sender-container': ['row'],
      '.iwt-message-sender': ['col-sm-12'],
      '.iwt-message': ['row'],
      '.iwt-message-vertical-spacer': ['visible-xs', 'col-xs-12'],
      '.iwt-message-text-wrapper': ['col-lg-9', 'col-lg-pull-3', 'col-sm-8', 'col-sm-pull-4', 'col-xs-12'],
      '.iwt-message-time-container': ['col-lg-3', 'col-lg-push-9', 'col-sm-4', 'col-sm-push-8', 'col-xs-12'],
      '.iwt-message-time': ['sm-pull-right'],
      '.iwt-send-on-enter-container': ['col-xs-12', 'col-sm-9', 'col-sm-push-3'],
      '.iwt-send-on-enter-inner-container': ['sm-pull-right'],
      '.iwt-chat-container-panel-bottom': ['row'],
      '.iwt-chat-exit-button-container': ['col-xs-12', 'col-sm-3', 'col-sm-pull-9'],
      '.iwt-compose-message-panel': ['row'],
      '#iwt-compose-message-textarea': ['col-xs-8', 'col-sm-10'],
      '.iwt-send-button': ['btn', 'btn-lg', 'btn-primary', 'col-xs-4', 'col-sm-2'],
      '.iwt-exit-button': ['btn', 'btn-lg', 'btn-danger']
    };

    // Language-specific maps

    // In Spanish, the text for "Email Address" wraps to a second line in "sm", so widen the left column of
the forms.
    this._map_es = this._generateGridMapping(
    {
      'sm_label': 6,
      'md_label': 4,
      'lg_label': 3
    });

    this._map_ja = this._generateGridMapping({
      'sm_optional_label': 3,
      'lg_optional_label': 2
    });
```

```
      this._map_ru = this._generateGridMapping({
        'sm_label': 4,
        'lg_label': 3,
        'lg_optional_label': 2
      });
      $j.extend(this._map_ru,
      {
        '.iwt-print-div': ['col-lg-3', 'col-lg-push-9', 'col-sm-5', 'col-sm-push-7'],
        '.iwt-participants-panel-list-container': ['col-lg-9', 'col-lg-pull-3', 'col-sm-7', 'col-sm-pull-5']
      });

      this._map_tr =
      {
        '.iwt-optional-label': ['col-sm-2', 'col-sm-push-7', 'col-lg-2', 'col-lg-push-8'],
        '.iwt-textbox-container-optional': ['col-sm-7', 'col-sm-pull-2', 'col-lg-8', 'col-lg-pull-2']
      };

      var languageCode = localization.LanguageCode;
      languageCode = languageCode.replace("-", "_");
      this._languageMap = this["_map_"+languageCode];
    },

  /**
   * Takes an IWT CSS class/id, and returns an array of the Bootstrap class(es) that should also be used on
the specified elements.
   *
   * Example:
   * getExtraCssClassesFor('.iwt-form-tabs') returns ['nav', 'nav-tabs']
   *
   * Since the return value may only specify CSS class(es), not id(s), the "." will not be present.
   * Note that more complex CSS selectors, such as ".iwt-form-tabs-container > .iwt-form-tabs" may not be
used.
   */
  getExtraCssClassesFor : function(iwtCssSelector)
  {
    if (this._languageMap && this._languageMap[iwtCssSelector])
    {
      return this._languageMap[iwtCssSelector];
    }
    return this._map[iwtCssSelector] || null;
  },

  _generateGridMapping : function(params) {
    var sm_label = params.sm_label || this._default_sm_label;
    var sm_optional_label = params.sm_optional_label || this._default_sm_optional_label;
    var sm_textbox = 12 - (sm_label);
    var sm_optional_textbox = 12 - (sm_label + sm_optional_label);
    var md_label = params.md_label || this._default_md_label;
    var md_optional_label = params.md_optional_label || this._default_md_optional_label;
    var md_textbox = 12 - (md_label);
    var md_optional_textbox = 12 - (md_label + md_optional_label);
    var lg_label = params.lg_label || this._default_lg_label;
    var lg_optional_label = params.lg_optional_label || this._default_lg_optional_label;
    var lg_textbox = 12 - (lg_label);
    var lg_optional_textbox = 12 - (lg_label + lg_optional_label);
    return {
      '.iwt-form-label': ['col-sm-'+sm_label, 'col-md-'+md_label, 'col-lg-'+lg_label, 'control-label'],
      '.iwt-optional-label': ['col-sm-'+sm_optional_label, 'col-sm-push-'+sm_optional_textbox, 'col-md-
'+md_optional_label, 'col-md-push-'+md_optional_textbox, 'col-lg-'+lg_optional_label, 'col-lg-push-
'+lg_optional_textbox],
      '.iwt-textbox-container': ['col-sm-'+sm_textbox, 'col-md-'+md_textbox, 'col-lg-'+lg_textbox],
      '.iwt-textbox-container-optional': ['col-sm-'+sm_optional_textbox, 'col-sm-pull-'+sm_optional_label,
'col-md-'+md_optional_textbox, 'col-md-pull-'+md_optional_label, 'col-lg-'+lg_optional_textbox, 'col-lg-pull-
'+lg_optional_label],
      '.iwt-form-field-no-label': ['col-sm-'+sm_textbox, 'col-sm-offset-'+sm_label, 'col-md-'+md_textbox,
'col-md-offset-'+md_label, 'col-lg-'+lg_textbox, 'col-lg-offset-'+lg_label],
      '.iwt-form-field-error': ['col-sm-'+sm_textbox, 'col-sm-offset-'+sm_label, 'col-md-'+md_textbox, 'col-
md-offset-'+md_label, 'col-lg-'+lg_textbox, 'col-lg-offset-'+lg_label],
      '.iwt-form-button-div': ['col-sm-'+sm_textbox, 'col-sm-offset-'+sm_label, 'col-md-'+md_textbox, 'col-md-
offset-'+md_label, 'col-lg-'+lg_textbox, 'col-lg-offset-'+lg_label]
```

```
      };
  }
});
```

RegistrationFormPanel

RegistrationFormPanel class

This is the panel that is shown when the user clicks "Register New Account". It allows them to create an account within tracker, that may then be used to create authenticated chats and/or callbacks.

Do not instantiate this class directly. Use
webservices.CustomizationFactoryRegistry.create_instance(webservices.CustomizableFactoryTypes.RegistrationFormPanel, args)
- args shall be a JSON object with the following properties:
- registrationManager: An instance of a class derived from RegistrationManagerBase.
- registerFormContainer: The Panel that contains this registration form. Must have a showRegisterForm() method.
- registrationCallback: The function to call once the registration attempt is complete (if it succeeds). May be null.
- form: An existing form to add the registration formfields to. May be null, in which case a new form will be created.

```
ui._Internal._DefaultRegistrationFormPanel = Class.create(ui.FormPanelBase,
{
 /**
  * Constructor
     *
     * @param args A Javascript object with the following members:
  * registrationManager An instance of a class derived from RegistrationManagerBase.
  * registerFormContainer The Panel that contains this registration form.  Must have a showRegisterForm()
method.
  * registrationCallback The function to call once the registration attempt is complete (if it succeeds).  May
be null.
  * form An existing form to add the registration formfields to.  May be null, in which case a new form will
be created.
  */
   initialize : function($super, args)
   {
     if(args.form)
     {
         this._form = args.form;
     }
     else
     {
         this._form = this.createDefaultForm();
     }

     $super(args.registerFormContainer, localization.Register, 'iwt-register-form-panel');

     this._registrationManager = args.registrationManager;
     this._externalRegistrationCallback = args.registrationCallback;
   },

 /**
  * Destructor
  */
   destroy : function()
   {
     this._registrationManager = null;

     ui.FormPanelBase.prototype.destroy.call(this);
   },

   // public methods

 /**
  * Called when this form receives focus.  Does some UI alignment, and delegates focus to the top field in the
form.
  */
   focus : function()
   {
```

```
      // username textbox should always be here, but just in case
      if(this._authenticatedIdentifierTextBox)
      {
        try
        {
          this._authenticatedIdentifierTextBox.focus();
        } catch (e)
        {
          common.Debug.traceWarning('Could not focus the correct textbox.');
        }
      }
    },

/**
 * Resets the form to its original state.
 */
    reset : function()
    {
      ui.FormPanelBase.prototype.reset.call(this);
      this._clearTextboxIfAvailable(this._authenticatedIdentifierTextBox);
      this._clearTextboxIfAvailable(this._authenticatedCredentialsTextBox);
      this._clearTextboxIfAvailable(this._confirmPasswordTextBox);
      this._clearTextboxIfAvailable(this._firstNameTextBox);
      this._clearTextboxIfAvailable(this._middleNameTextBox);
      this._clearTextboxIfAvailable(this._lastNameTextBox);
      this._clearTextboxIfAvailable(this._departmentTextBox);
      this._clearTextboxIfAvailable(this._companyTextBox);
      this._clearTextboxIfAvailable(this._jobTitleTextBox);
      this._clearTextboxIfAvailable(this._assistantNameTextBox);
      this._clearTextboxIfAvailable(this._homeStreetAddressTextBox);
      this._clearTextboxIfAvailable(this._homeCityTextBox);
      this._clearTextboxIfAvailable(this._homeStateTextBox);
      this._clearTextboxIfAvailable(this._homePostalCodeTextBox);
      this._clearTextboxIfAvailable(this._homeCountryTextBox);
      this._clearTextboxIfAvailable(this._homeEmailTextBox);
      this._clearTextboxIfAvailable(this._homePhoneTextBox);
      this._clearTextboxIfAvailable(this._homePhone2TextBox);
      this._clearTextboxIfAvailable(this._homeFaxTextBox);
      this._clearTextboxIfAvailable(this._homePagerTextBox);
      this._clearTextboxIfAvailable(this._homeMobileTextBox);
      this._clearTextboxIfAvailable(this._homeUrlTextBox);
      this._clearTextboxIfAvailable(this._businessStreetAddressTextBox);
      this._clearTextboxIfAvailable(this._businessCityTextBox);
      this._clearTextboxIfAvailable(this._businessStateTextBox);
      this._clearTextboxIfAvailable(this._businessPostalCodeTextBox);
      this._clearTextboxIfAvailable(this._businessCountryTextBox);
      this._clearTextboxIfAvailable(this._businessEmailTextBox);
      this._clearTextboxIfAvailable(this._businessPhoneTextBox);
      this._clearTextboxIfAvailable(this._businessPhone2TextBox);
      this._clearTextboxIfAvailable(this._businessFaxTextBox);
      this._clearTextboxIfAvailable(this._businessPagerTextBox);
      this._clearTextboxIfAvailable(this._businessMobileTextBox);
      this._clearTextboxIfAvailable(this._businessUrlTextBox);
      this._clearTextboxIfAvailable(this._assistantPhoneTextBox);
      this._clearTextboxIfAvailable(this._remarksTextBox);
      this.enableFormFields();
    },

    createDefaultForm : function()
    {
      common.Debug.traceMethodEntered("RegistrationFormPanel.createDefaultForm()");
      var section = new ui.FormSection(localization.Account)
                  .addFieldByFieldType(ui.FormFieldTypes.Username)
                  .addFieldByFieldType(ui.FormFieldTypes.Password)
                  .addFieldByFieldType(ui.FormFieldTypes.ConfirmPassword);
      frm = new ui.Form([section]);

      common.Debug.traceMethodExited("RegistrationFormPanel.createDefaultForm()");
      return frm;
    },
```

```
enableFormFields : function()
{
  this._enableElement(this._authenticatedIdentifierTextBox);
  this._enableElement(this._authenticatedCredentialsTextBox);
  this._enableElement(this._confirmPasswordTextBox);
  this._enableElement(this._firstNameTextBox);
  this._enableElement(this._middleNameTextBox);
  this._enableElement(this._lastNameTextBox);
  this._enableElement(this._departmentTextBox);
  this._enableElement(this._companyTextBox);
  this._enableElement(this._jobTitleTextBox);
  this._enableElement(this._assistantNameTextBox);
  this._enableElement(this._homeStreetAddressTextBox);
  this._enableElement(this._homeCityTextBox);
  this._enableElement(this._homeStateTextBox);
  this._enableElement(this._homePostalCodeTextBox);
  this._enableElement(this._homeCountryTextBox);
  this._enableElement(this._homeEmailTextBox);
  this._enableElement(this._homePhoneTextBox);
  this._enableElement(this._homePhone2TextBox);
  this._enableElement(this._homeFaxTextBox);
  this._enableElement(this._homePagerTextBox);
  this._enableElement(this._homeMobileTextBox);
  this._enableElement(this._homeUrlTextBox);
  this._enableElement(this._businessStreetAddressTextBox);
  this._enableElement(this._businessCityTextBox);
  this._enableElement(this._businessStateTextBox);
  this._enableElement(this._businessPostalCodeTextBox);
  this._enableElement(this._businessCountryTextBox);
  this._enableElement(this._businessEmailTextBox);
  this._enableElement(this._businessPhoneTextBox);
  this._enableElement(this._businessPhone2TextBox);
  this._enableElement(this._businessFaxTextBox);
  this._enableElement(this._businessPagerTextBox);
  this._enableElement(this._businessMobileTextBox);
  this._enableElement(this._businessUrlTextBox);
  this._enableElement(this._assistantPhoneTextBox);
  this._enableElement(this._remarksTextBox);
},

disableFormFields : function()
{
  this._disableElement(this._authenticatedIdentifierTextBox);
  this._disableElement(this._authenticatedCredentialsTextBox);
  this._disableElement(this._confirmPasswordTextBox);
  this._disableElement(this._firstNameTextBox);
  this._disableElement(this._middleNameTextBox);
  this._disableElement(this._lastNameTextBox);
  this._disableElement(this._departmentTextBox);
  this._disableElement(this._companyTextBox);
  this._disableElement(this._jobTitleTextBox);
  this._disableElement(this._assistantNameTextBox);
  this._disableElement(this._homeStreetAddressTextBox);
  this._disableElement(this._homeCityTextBox);
  this._disableElement(this._homeStateTextBox);
  this._disableElement(this._homePostalCodeTextBox);
  this._disableElement(this._homeCountryTextBox);
  this._disableElement(this._homeEmailTextBox);
  this._disableElement(this._homePhoneTextBox);
  this._disableElement(this._homePhone2TextBox);
  this._disableElement(this._homeFaxTextBox);
  this._disableElement(this._homePagerTextBox);
  this._disableElement(this._homeMobileTextBox);
  this._disableElement(this._homeUrlTextBox);
  this._disableElement(this._businessStreetAddressTextBox);
  this._disableElement(this._businessCityTextBox);
  this._disableElement(this._businessStateTextBox);
  this._disableElement(this._businessPostalCodeTextBox);
  this._disableElement(this._businessCountryTextBox);
```

```
        this._disableElement(this._businessEmailTextBox);
        this._disableElement(this._businessPhoneTextBox);
        this._disableElement(this._businessPhone2TextBox);
        this._disableElement(this._businessFaxTextBox);
        this._disableElement(this._businessPagerTextBox);
        this._disableElement(this._businessMobileTextBox);
        this._disableElement(this._businessUrlTextBox);
        this._disableElement(this._assistantPhoneTextBox);
        this._disableElement(this._remarksTextBox);
    },

// private methods

    _buildInnerPanel : function(prefix)
    {
      var container = this.createElement('div', null, {'class': 'iwt-form-container'});
      var sections = this._form.get_sections();
      for(var i = 0; i
```

Change Log

The following table lists the changes to the *Interaction Web Tools Developer's Guide* since its initial release.

Date	Changes
21-October-2011	Initial release version.
10-January-2012	- Added section on adding fields to the chat registration form. - Made a few other minor tweaks. - Updated copyrights and other standard text.
25-January-2012	- Added section on adding custom information to chat/callback. - Corrected capitalization of Bootloader.js in several locations.
30-July-2013	- Corrected a code line in the "Reconnect" section. - Updated code in Appendices A and B. - Did other general cleanup.
05-August-2013	Updated code for JSON, discussed routing contexts, and made other miscellaneous updates.
08-August-2013	Added 2,000-character subject field limit info to "Create a Callback."
09-October-2013	Added a new section with HTTP header information.
02-December-2013	Added "Sending and Receiving Web Pages Using Handlers." The content had been in the Web Services Technical Reference. Modified for inclusion in the current document.
18-March-2014	Updated "Embedding Within Another Web Page," "Chats And Callbacks to Multiple Queues," "Creating a Post-Chat Page," and "Adding Custom Information to a Chat" for some JavaScript code changes.
21-May-2014	Added information about transcript features.
12-June-2014	Added section about Interaction Mobilizer.
08-August-2014	Updated documentation to reflect changes required in the transition from version 4.0 SU# to CIC 2015 R1, such as updates to product version numbers, system requirements, installation procedures, references to Interactive Intelligence Product Information site URLs, and copyright and trademark information.
25-January-2014	Added information about developing responsive UIs for mobile devices and other enhancements.
05-October-2015	Updated cover page and copyright information.
15-December-2015	Moved information about Agent Search Page from *Interaction Web Tools Technical Reference* to this document.
21-December-2015	Removed "Appendix B: How to Set and Use JavaScript" Tracing because that information is already in *Interaction Web Tools Technical Reference*.

29-December-2015	• Moved "Designating which Agents Appear on the Agent Search Page" to *Interaction Web Tools Technical Reference*, because it's configuration, not customization. • Reorganized content. Moved "Customizing the Agent Search Page" after "Sending and Receiving Web Pages Using Handlers" because it is a specific example of using a handler.
24-February-2017	Corrected example in "Change Default Send on Enter Behavior" section.
31-May-2017	Added "Appendix B: Customization Points"
06-September-2017	• Rebranded document to apply Genesys terminology. • Updated cover page, copyright, trademark pages.
02-February-2018	Interaction Mobilizer is a deprecated product. Removed references to Mobilizer SDK from this publication. Specifically, removed "Using the Interaction Mobilizer SDK", "Example: Mobilizer with iOS Apps", and "Example: Mobilizer with Phonegap Apps".
11-March-2019	Corrected description of SystemIVRWorkgroupQueueAlert in Web-related Handlers.
20-June-2019	Reorganized the content only, which included combining some topics and deleting others that just had an introductory sentence such as, "In this section...".
26-September-2019	Added to the Making Customizations topic. Note: The location of the customization files depends on the selections the system administrator made when installing Interaction Web Tools.