Instant Queue Manager V4 Developer Guide

November 6, 2013

Rev 06_NOV_2013

Copyright and Disclaimer

Trademarks

Contact Information

See our Web site for Customer Support information.

http://www.instant-tech.com/

Contents

Overview

Instant Queue Manager provides a set of services, and extension points, in order to provide developers with the ability to customize and extend many areas of the application. While it is quite possible to deploy Instant Queue Manager 'out of the box', Instant has provided the ability to extend Queue Manager in three main areas:

1. Server side extensions and libraries that provide an event system as well as Java, Groovy, and Web Service APIs. This includes server based events, server based java and groovy plugins, and a web service API.
2. Web client extensions and branding, with the ability to post custom data and start web based conversations with the server.
3. Custom 'expert' panel extensions to embed within the expert's Sametime chat interface in order to integrate an embedded web panel with an inbound chat session – attached to the expert's chat session.

At a high level, it should be fairly sifsdfmple for a developer to design a system that provides a custom web chat experience, post information about the customer to the Queue, and then provide all of the customer's information to the expert as the expert accepts the inbound chat session. With a bit more development, the Queue server may be extended with custom modules that 'lookup' additonal information on the customer (possibly from a CRM system) and automatically create and/or close a ticket in a CRM system.

Typical integration scenarios include:

- Pass custom values (such as account number, location, etc..) from the web client to the server, and then to the expert as the inbound customer is routed
- Automatically create a ticket in the enterprise CRM system as inbound chat requests are posted to a queue for routing
- Apply a 'metatag' or custom properties (such as VIP) to inbound custom sessions – so they may be routed to an appropriate queue or collection of experts
- Register for a set of server level events in order to provide additional internal lookups on waiting customers – and then apply these values and custom properties as the chat session is routed to an expert
- Create a fully customized chat IM conversation – possibly as a front end to an existing knowledge base or other helpful service.
- Integrate Queue Manager with an internal phone system – so that they expert may be 'disabled' automatically if they are engaged in a queue conversation.

At the server level, Queue Manager is a Java 'Spring' application with the ability to load custom Java and Groovy based modules, a defined event signalling mechanism, and a web service API. The server has been designed to enable deep integration with existing enterprise and/or cloud based applications. Scenarios for server based integration include the ability to raise, and catch, internal notifications on queue status (for example, raise an event when a customer enters a queue), the ability to appliy custom values to an inbound seeker session, and the ability to to interact with the seeker via custom Java or Groovy modules.

Other possible server based extensions include the ability to interact with external phone systems, via the web service API, the ability to external systems to enable or disable queues and experts, and the

ability to post information (such as tickets and ticket status) to external CRM or ticketing systems. Instant Queue Manager exposes a set of web service calls which allows developers to interact with the Queue Manager engine in real time. These web serivces APIs allow developers to fetch queue queue definitions, modify or set expert properties, and optionally broadcast messages.

As customers, or seekers, interact with the system from a web portal or web page, the ability to customize and embed the web based chat interface is also quite important. Queue Manager provides a flexible web chat client with the ability to brand the client, pass custom values (JSON posts) to the Queue Manager server, and the ability to 'skin' the web client. Typical scenarios include the ability for a web based customer to start a chat session with the a 'queue', the ability to post custom information about the customer to the queue, the ability for the web UI to query the status of the queue, and the ability to provide custom branding and integration with the web client.

As a development team, we hope you find this document helpful and we also hope that the integration points within Queue Manager provide you with the ability to create useful integration with your existing systems and platforms. We are striving to provide a rich platform for integration and we understand that Queue Manager exists within a system of systems – and that integration with existing enterprise applications is not only important, but a critical function of our customer chat management and routing platform.

We hope you find this guide useful. Please contact us with any questions, comments, or ideas.

You can always send comments to our email: support@instant-tech.com or contact us on Twitter @TeamInstant

Thanks

The folks at Instant Technologies

Web Service Calls

This section provides a comprehensive list of web service calls exposed by Instant Queue Manager V4 engine.

WSDL URL for a typical Queue Manager deployment is as follow:
http://<ApacheServer>/ITFramework/ITApplicationManagerPort?wsdl

Get list of databases associated with IQMV4 engine

This web service call allows you to get list of databases from which Queue engine is fetching queue definitions. Since the engine can load queue definitions from multiple databases it is necessary to specify database URL in subsequent method calls.

Function: getAllAppURLs()

Parameters: None

Output:

```
<AppURLs>
        <URL>http://<dominoserver>/sales/itqconnect.nsf</URL>
        <URL>http://<dominoserver>/support/itqconnect.nsf</URL>
</AppURLs>
```

Note: Database URL fetched using this function call is to be passed in all subsequent method calls.

Get list of queues

This web service call allows you to get list of queues that are currently running under IQMV4 engine along with a brief summary of total number of waiting and connected seekers.

Function: getAllMonitoringQueues(String databaseURL)

Parameters:
databaseURL: This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

Output:

```
<MonitoringQueues>
  <Queue>
        <LoginName>fastclaim</LoginName>
        <QueueName>Fast Claim</QueueName>
        <QueueStatus>Online</QueueStatus>
        <Description>Test Queue</Description>
        <StartTime>Tue Oct 5 2013 04:18:50 AM</StartTime>
        <SeekersCount>0</SeekersCount>
        <ExpertsCount>11</ExpertsCount>
        <OnlineExpertsCount>3</OnlineExpertsCount>
        <ConnectedExpertsCount>0</ConnectedExpertsCount>
        <ConnectedSeekers>0</ConnectedSeekers>
        <WaitingSeekers>0</WaitingSeekers>
        <MaxWaitingSeeker/>
        <WaitingSince>0</WaitingSince>
   </Queue>
</MonitoringQueues>
```

Get list of seekers

This method allows you to get list of all seekers that are currently present in Queue. A seeker might be either in waiting state or connected with an expert.

Function: getSeekers(String databaseURL, String queueName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

queueName: This refers to the Sametime ID of an Queue like: CN=QueueName, OU=Dev

Output:

```
<MonitoringSeekers>
 <Seeker>
        <SeekerName>John Row</SeekerName>
        <WebVisitorName> John Row </WebVisitorName>
        <SeekerState>Waiting</SeekerState>
        <QueueName>fastclaim</QueueName>
        <InitialRequest>Need help with my policy: XY125</InitialRequest>
        <ArrivalTime>Tue Oct 5 2013 04:49:51 AM</ArrivalTime>
        <ExpertAssigned>na</ExpertAssigned>
        <WaitingTime>24 s</WaitingTime>
        <ConnectingTime>na</ConnectingTime>
        <WaitingSince>24</WaitingSince>
        <SeekerDisplayName> John</SeekerDisplayName>
 </Seeker>
 <Seeker>
        <SeekerName>Bill Bolts</SeekerName>
        <WebVisitorName> Bill Bolts </WebVisitorName>
        <SeekerState>Connected</SeekerState>
        <QueueName>fastclaim</QueueName>
        <InitialRequest>Need help with my refund</InitialRequest>
        <ArrivalTime>Tue Oct 5 2013 06:49:51 AM</ArrivalTime>
        <ExpertAssigned>Suzy Sparks</ExpertAssigned>
        <WaitingTime>27 s</WaitingTime>
        <ConnectingTime>60 s</ConnectingTime>
        <WaitingSince>27</WaitingSince>
        <SeekerDisplayName> Bill</SeekerDisplayName>
 </Seeker>
</MonitoringSeekers>
```

Get list of experts and their state

This method allows you to get a list of all users that are member/experts in a Queue.

Function: getExperts(String databaseURL, String queueName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

queueName: This refers to the Sametime ID of an Queue like: CN=QueueName, OU=Dev

Output:

```
<MonitoringExperts>
        <Expert>
                <ExpertName>CN=FirstName LastName/OU=US/O=dev</ExpertName>
                <ExpertDescription>NA</ExpertDescription>
                <QueueName>Fast Claim</QueueName>
                <ExpertState>free|connected</ExpertState>
                <AssignedSeekers>na|seekerID</AssignedSeekers>
        </Expert>
        <Expert>
                …
        </Expert>
</MonitoringExperts>
```

Get seeker properties

This method allows you to retrieve meta data associated with a seeker like their IP address, initial query, email address, browser details etc…

Function: getSeekerProperties(String databaseURL,String queueName,String seekerID)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

queueName: This refers to the Sametime ID of an Queue like: CN=QueueName, OU=Dev

seekerID: This refers to the Sametime ID of a seeker like: *CN=FirstName FullName, OU=Web*

Output:

```
<SeekerDetail>
        <IpAddress>NA</IpAddress>
        <SametimeId>CN=John Row/OU=Web</SametimeId>
        <SametimeName> John Row</SametimeName>
        <InterviewStartNode>NA</InterviewStartNode>
        <FirstName/>
        <LastName/>
        <CustomerEmail/>
        <SendTrans/>
        <Question>Need Help</Question>
        <visitorTokenId>NA</visitorTokenId>
        <BrowserName>NA</BrowserName>
        <BrowserVersion>NA</BrowserVersion>
        <BrowserOS>NA</BrowserOS>
        <CommentMsg>NA</CommentMsg>
        <ChatStartTime>Oct 5 2013 05:16:15 AM</ChatStartTime>
        <IframeParent>NA</IframeParent>
        <Name/>
        <ComputerID/>
        <Location/>
        <Extension/>
 </SeekerDetail>
```

Get seeker chat conversation text

This method allows you to get chat conversation text in realtime for the specified seeker.

Function: getChatConversation(String databaseURL, String seekerID, String queueName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

seekerID: This refers to the Sametime ID of a seeker like: *CN=FirstName FullName, OU=Web*

queueName: This refers to the Sametime ID of an Queue like: CN=QueueName, OU=Dev

Output:

NA (if no data is available)

Or

Actual Conversation Text

Get queue statistics

This method allows you to retrieve various queue counters like total number of inbound requests received by the queue, number of experts registered/available in queue etc..

Function: getQueueStatistics(String databaseURL, String queueName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

queueName: This refers to the Sametime ID of an Queue like: CN=QueueName, OU=Dev

Output:

```
<QueueStatistics>
        <InboundRequests>400</InboundRequests>
        <AvailableAgents>15</AvailableAgents>
        <RegisteredAgents>20</RegisteredAgents>
        <AcceptedChats>326</AcceptedChats>
</QueueStatistics>
```

Get Expert state across all queues

This method allows you to get expert state in Queue engine. An expert might be either in “Available” or “Unavailable” state.

Function: getExpertStatusAcrossQueues(String databaseURL, String expertID)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

expertID: This refers to the Sametime ID of expert like: *CN=FirstName FullName, OU=Support*

Output:

```
<Queues>
	<Queue>
		<QueueName>Pharmacy</QueueName>
		<QueueLogin>pharmacy</QueueLogin>
		<ExpertSametimeIMAddress>vgtest user97</ExpertSametimeIMAddress>
		<ExpertPersonIMAddress>VGTest User97</ExpertPersonIMAddress>
		<ExpertState>NOTAVAILABLE</ExpertState>
		<ExpertStatus>Offline</ExpertStatus>
		<ActiveConversation>0</ActiveConversation>
		<TotalConversation>0</TotalConversation>
		<Group>NA</Group>
		<Priority>3</Priority>
		<ExpertQueueState>Disabled</ExpertQueueState>
	</Queue>
</Queues>
```

Set Expert state across all queues

This method allows you to set expert state in Queue engine. An expert can be either in “Available” or “Unavailable” state.

Function: setExpertStatusAcrossQueues(String appURL,String expertID,String state)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

expertID: This refers to the Sametime ID of expert like: *CN=FirstName FullName, OU=Support*

state: NOTAVAILABLE or FREE

Output:

True

Or

False

Set Expert state in a queue

This method allows you to set expert state in a Queue. An expert can be either in “Available” or “Unavailable” state.

Function: setExpertState(String databaseURL, String queueName, String expertID, String state)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

queueName: This refers to the Sametime ID of Queue like: CN=QueueName, OU=Dev

expertID: This refers to the Sametime ID of expert like: *CN=FirstName FullName, OU=Support*

state: NOTAVAILABLE or FREE

Output:

True

Or

False

Start Queue/Interview

This method allows you to start Queue/Interview that might be registered with the IQMV4 engine but not running.

Function: startBot(String databaseURL, String applicationName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

applicationName: This refers to the Sametime ID of Queue or Interview like: CN=QueueName, OU=Dev

Output:

True

Or

False

Stop Queue/Interview

This method allows you to stop Queue/Interview that might be registered and running under IQMV4 engine.

Function: stopBot(String databaseURL, String ApplicationName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

applicationName: This refers to the Sametime ID of Queue or Interview like: CN=QueueName, OU=Dev

Output:

True

Or

False

Reload all Queue/Interview definitions

This method allows you to instruct IQMV4 engine to reload/sync all queue and interview definitions by fetching them again from the database.

Function: reloadBots(String databaseURL)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

Output:

True

Or

False

Reload Queue/Interview definition

This method allows you to instruct IQMV4 engine to reload/sync definition for the specified queue/interview.

Function: updateMonitoringQueues(String databaseURL, String applicationName)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

applicationName: This refers to the Sametime ID of Queue or Interview like: CN=QueueName, OU=Dev

Output:

True

Or

False

Get IQMV4 runtime engine version number

This method allows you to get IQMV4 runtime version number.

Function: getVersion(String databaseURL)

Parameters:
databaseURL**:** This refers to the URL of the database which contains queue configuration fetched using getAllAppURLs(), like: http://<dominoserver>/sales/itqconnect.nsf

Output:

4.2.XXXXX

Setting up Web Client

IQMV4 installer installs the web client into following directory under Apache Tomcat server:

<Apache Tomcat Folder>/Tomcat 7.0/webapps/ITFramework/itchat

HTTP URL for accessing web client is as follows:

http://<apacheServerFQDN>/ITFramework/itchat/chatui5.html?userName=QueueLoginName

Configuring Web Client

To configure web client you will need to modify settings.js file present in the itchat folder.

settings.js looks like the following screen shot:

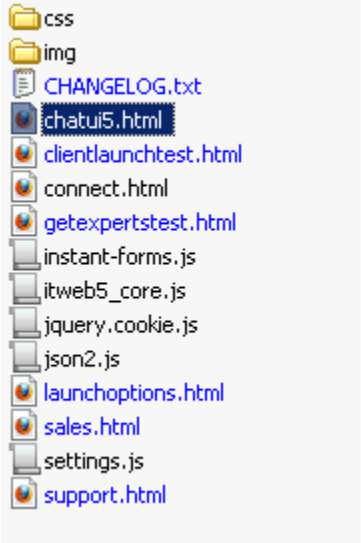
```
var _hostPrefix = 'http://';
var _myHostURL = "chatui.instant-tech.com/ITFramework";
var _mySTServerLocation = "stamazon.instant-tech.com";
var _servletPrefix = '/webclient';
```

In the file you will need to specify value for following properties:

1. _myHostURL: This should contain FQDN for server running Apache Tomcat Server.
2. _mySTServerLocation: This should contain FQDN for server running IBM Sametime Server

Updating Styling CSS/Logo

Web Client folder contains following files:



To tweak client styling you will need to edit chatui5.html page and modify css files contained in css folder.

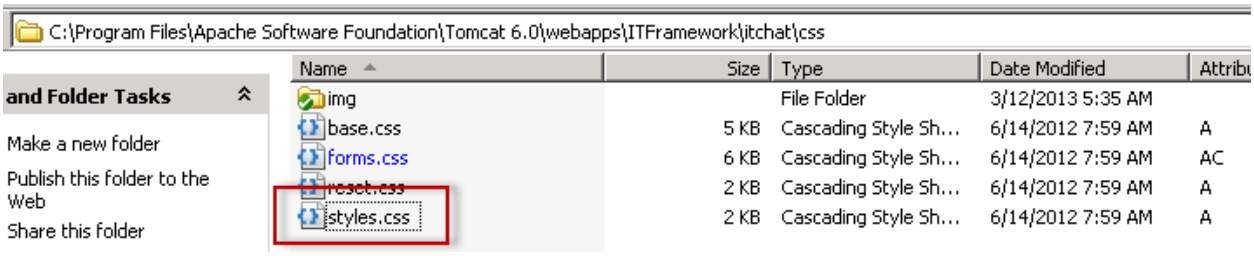
Updating logo

To update the logo you will need to edit "**chatui5.html**", look for image tag contained in the DIV implemting css class logo and specify URL to the newer logo file.

```
225 <body onload="startLoading();" onbeforeunload="event_WindowClose();">
226 <div id="container">
227     <div class="top">
228         <div class="logo">
229             <img src="css/img/logo-final.png" alt="Instant Technologies" />
230         </div>
231         <div class="details">
232             <!-- Details (user picture, queue details, etc) -->
233         </div>
234     </div>
235     <div class="clear"><!-- Clear --></div>
236     <div class="window" id="transcript">
237
238     </div>
239     <div class="clear"><!-- Clear --></div>
240     <div class="options">
241         <!-- <ul id="menu">
```

Updating chat window styling/colors

Chat window styling/colors are defined in “styles.css” present in css folder.



Style.css defines all css attributes used in “**chatui5.html**”. Contents of the css file look like the following screen shot:

```
/* This file contains easily customizeable elements of the chat window itself.
 * This is intended for quick image replacements, font adjustments, or color adjustments.
 * Structural or postional changes should be made from within the base.css file in this directory.
 */
body {
    background:url(../img/body-dark-pattern.png) repeat scroll;
    font-family: Tahoma,Arial,Helvetica,sans-serif;
}
div#container {
    background:url(../img/dark-bg-pattern.png) repeat scroll;
    border:1px #000 solid;
    background-attachment: fixed;
}
div.window {
    background:#ccc;
    border:1px #fff solid;
}
div.expert-response-wrapper {
    margin-top:5px;
    margin-left:5px;
    float:left;
    text-align:left;
    font-size:12px;
    width:250px;
}
div.expert-response-wrapper p.expert-response {
    width:250px;
}
```

Embedding Web Client into your site/portal

Embedding chat link to any web page is fairly easy. Take the following steps to embed chat link into your custom web page:

1. Open the webpage using any text editor
2. Add the following snippet to page header:

```
<script type="text/javascript" >
        function startchat(queue) {
        window.open("http://ApacheServerIP/ITFramework/itchat/chatui5.html?userName="
        + queue , "_blank", " resizable=yes,menubar=no,width=638,height=515");
        }
</script>
```

3. Add the following snippet to page body where you would want to display the chat link:

   Queue Display Name goes here

'Queue Login ID' in the above snippet needs to be replaced with the Queue Login Name.
You can customize the look and feel of the hyperlink as per your needs. Hyperlink needs to call the function: startchat('Queue Login ID') to initiate a chat when the hyperlink is clicked by the user.

A live demo of the customized chat links can be viewed at http://www.instant-tech.com/Interactive_Presence.cfm

Sending across seeker custom data to IQMV4 server

To collect and send across additional details i.e. **seeker meta-data** about a seeker to the IQMV4 server and experts in realtime you will need to format it as a JSON object. Structure of JSON data is as follows:

```
{"CRM":
        [{"key":"webVistor","value":"true"},
        {"key":"IpAddress","value":"192.168.8.10"},
        {"key":"STID","value":"John Doe"},
        {"key":"First_Name","value":"John"},
        {"key":"Last_Name","value":"Doe"},
        {"key":"CustomerEmail","value":"john@acme.com"},
        {"key":"Question","value":"please reset my email password"},
        {"key":"BrowserName","value":"IE"},
        {"key":"BrowserVersion","value":"8.0"},
        {"key":"BrowserOS","value":"Red Hat"},
        {"key":"location","value":"USA"},
        {"key":"SerialNumber","value":"S-0012"},
        {"key":"QueueID","value":"Tech Support"}]
}
```

JSON data should always contain **STID** key. **STID** refers to the seeker ID typically their **LDAP** Id or their common name.

All the other properties are **optional**. You can also send across custom properties like seeker subscription number, phone number, street address or any other details that will help expert serve seekers more efficiently.

JSON data with custom properties will look like the following snippet:

```
{"CRM":
        [{"key":"webVistor","value":"true"},
        {"key":"IpAddress","value":"192.168.8.10"},
        {"key":"STID","value":"John Doe"},
        {"key":"First_Name","value":"John"},
        {"key":"Last_Name","value":"Doe"},
        {"key":"CustomerEmail","value":"john@acme.com"},
        {"key":"Question","value":"please reset my email password"},
        {"key":"phoneNumber","value":"1-800-XXXX-XXXX"},
        {"key":"subscriptionId","value":"XYZ-1245"},
        {"key":"QueueID","value":"Tech Support"}]
}
```

JSON data needs to be sent to following URL before the chat session is started:
http://<TomcatServerFQDN>/ITFramework/servlet/CRMServlet?datajson=<JSON_Data_Goes_Here>

Server will return back server response as:

SUCCESS (if data was successfully posted)

Or

INVALID_STID (if the mandatory key STID is missing)

Or

ERROR (if JSON is malformed)

JavaScript function is as follows:

```
function postSeekerDetails(seekerMetaData) {
var url = 'http://<TOMCAT_SERVER>/ITFramework/servlet/CRMServlet?datajson=' + seekerMetaData;
        $.ajax({
                type: "POST",
                url: url,
                success: function(data) {
                        if (data == 'SUCCESS')
                        {
                                alert("data successfully posted");
                        }
                }
});
}
```

For sample JavaScript Code refer: **post_seekerproperties.html** file located in "**<Tomcat Install Folder>\webApps\ITFramework\samples\"** folder.

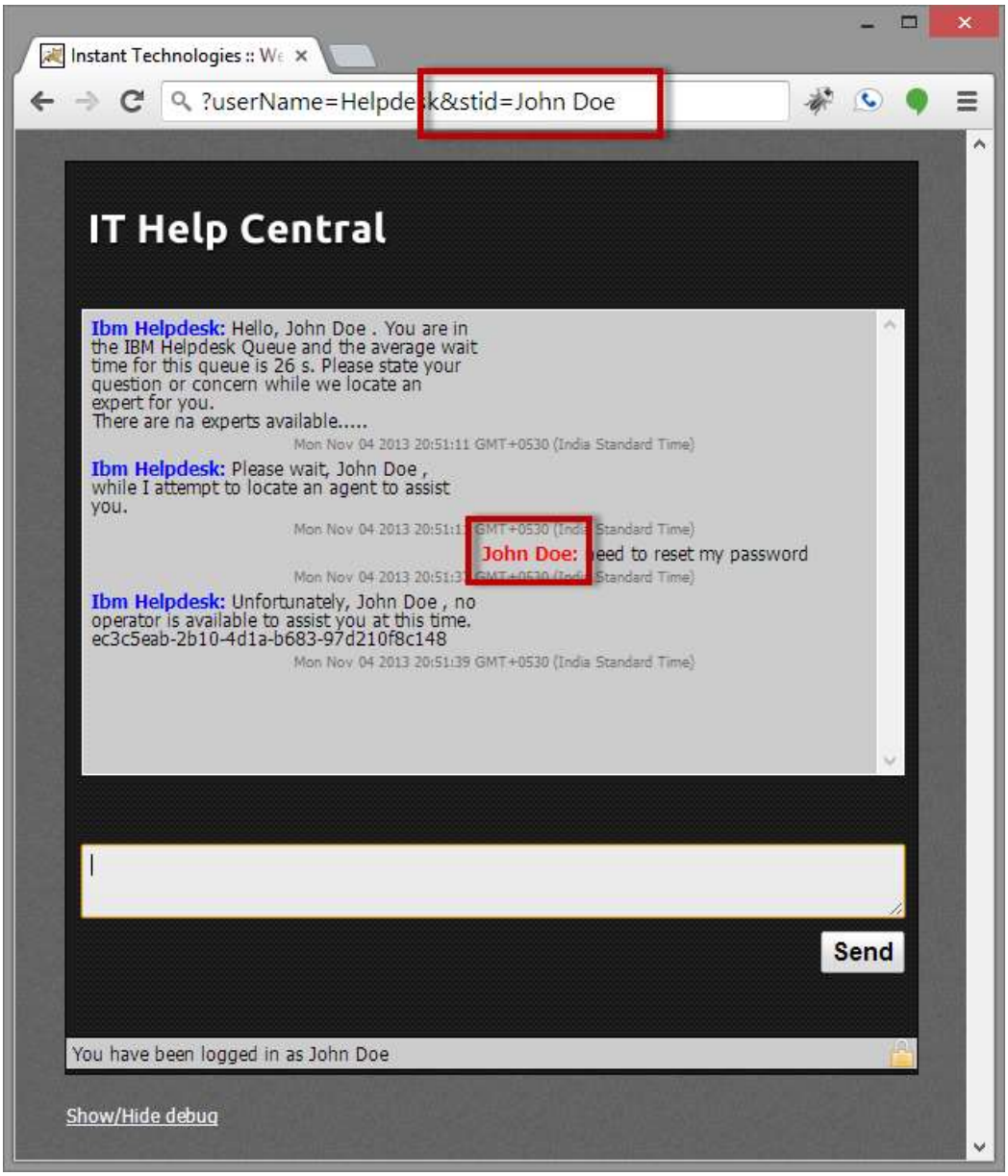
Sample demonstrates posting on individual custom seeker properties.

After seeker meta-data has been posted to IQMV4 engine, seeker **STID** should be used to launch web client. Web client will then refer seeker with the specified **STID**. For **anonymous users** their email address can be passed on to web client to personalize messages. If a value for STID is not specified web visitor will be assigned a **random Id** (like ITIM User 10, ITIM User 11) having format as ***ITIM User <Auto Incrementing Index>***

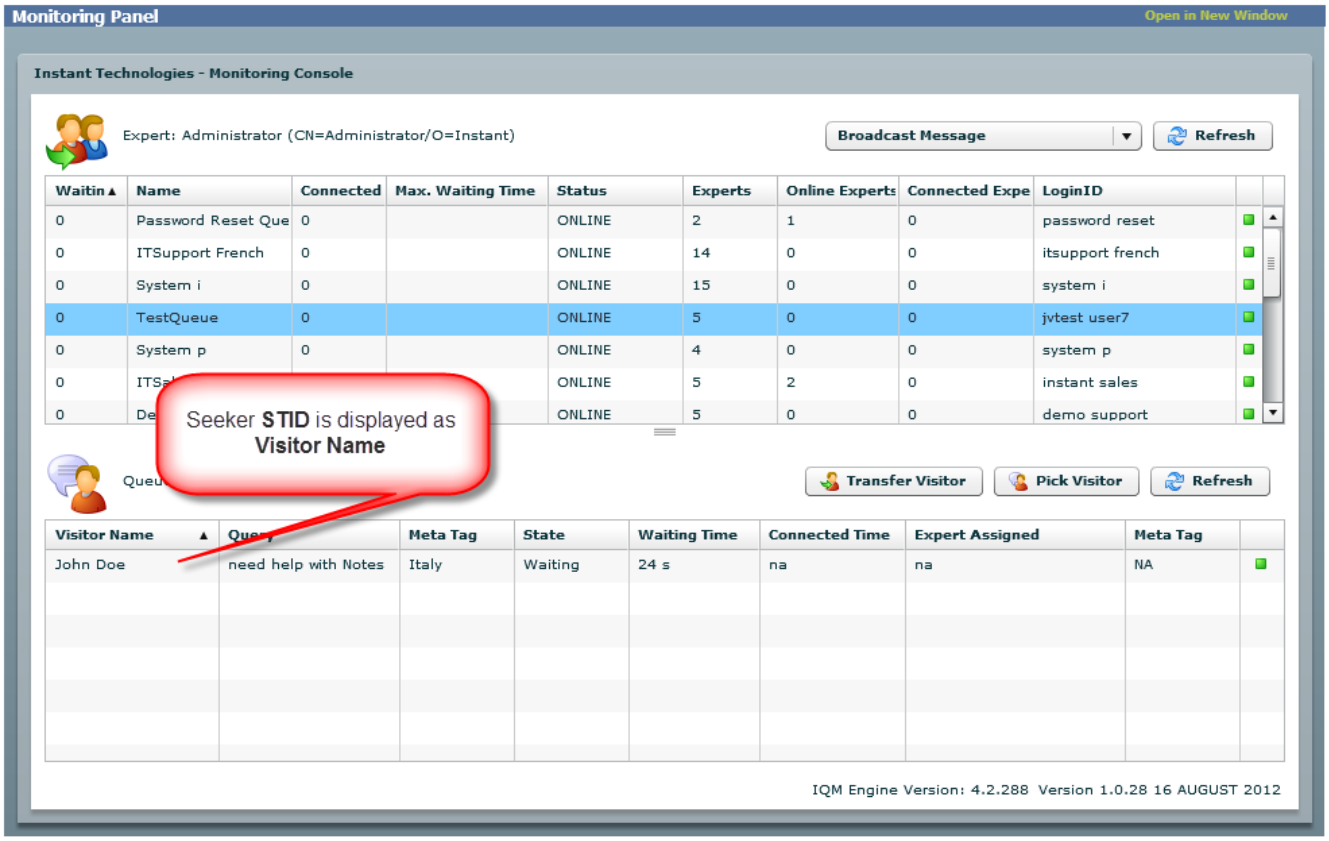
If seeker STID has been specified as **John Doe** in seeker meta data, it should be appended to web client URL as follows:

http://ApacheServerIP/ITFramework/itchat/chatui5.html?userName=<Queue_Name>&stid=John Doe

Seeker STID will be displayed as follows in web client:



In Monitoring Panel **Visitor Name** column shows seeker **STID** which has been posted as meta-data to IQMV4 engine.



Displaying Queue availability on your site/portal

After you have embedded web client into your site/portal you might want to activate the link only during your regular business hours. To enable this take following steps:

Query the IQMV4 engine to get Queue availability status. Following URL will return Queue status as JSON

http://<**TOMCAT_SERVER_FQDN**>/ITFramework/scheduler?appid=<Queue_Username>

Structure of JSON response is as follows:

[{"Status":**true**,
"NotAvailableURL":"http://TOMCAT_SERVER_FQDN/ITFramework/img/dnd.gif",
"AvailableURL":"http:// TOMCAT_SERVER_FQDN /ITFramework/img/active.gif",
"LoginName":"Sales Queue"}]

If status is **true** it will indicate that Queue is operating within regular business hours and available to serve incoming requests.

JavaScript function is as follows:

```
function getQueueStatus(queueId){
		$.ajax({
		type: "POST",
		url: 'http://<TOMCAT_SERVER>/ITFramework/scheduler?appid='+ queueId,
		success: function(data) {
			var json = $.parseJSON(data);
				$(json).each(function(i, queueData) {
					if (queueData.Status == true)
					{
						//Queue is in working hours
					}
					else
					{
						//Queue is in away state (outside of working hours)
					}
				}
			}
		});
	}
```

For sample JavaScript Code refer: **scheduler_presence.html** located in "**<Tomcat Install Folder>\ webApps\ITFramework\samples\"** folder.

If however Queue is set to operate in 24x7 mode you might want to Query Queue to verify if agents are available to serve incoming requests. Following URL will return count of available experts in Queue as JSON

http://<TOMCAT_SERVER>/ITFramework/SessionDetails?queueName=<Queue_Username>

Structure of JSON response is as follows:

[{"totalExperts":"19","**availableExperts**":"1"}]

If there are available experts then you can activate the chat link.

JavaScript function is as follows:

```
function getQueueStatus(queueId){
		$.ajax({
		type: "POST",
		url: 'http://<TOMCAT_SERVER>/ITFramework/SessionDetails?queueName='+ queueId,
		success: function(data) {
			var json = $.parseJSON(data);
				$(json).each(function(i, queueData) {
					if (queueData.availableExperts > 0)
					{
						//Queue is active
					}
					else
					{
						//Queue is in not available, all experts are busy or away
					}
				}
			}
		});
	}
```

For sample JavaScript Code refer: **expert_presence.html** located in "**<Tomcat Install Folder>\ webApps\ITFramework\samples\"** folder.

Sending seeker custom data to experts in notification messages

Seeker meta-data can be sent across to experts in IM notification messages which the Queue sends across to experts prompting them to pick up a conversation.

Steps are as follows:

1. Edit a Queue definition
2. Go to text tab
3. Locate the message "**Notify Operator**" (variables can be used in any text resource).
4. In the message you can access seeker custom data by specifying it in following format:

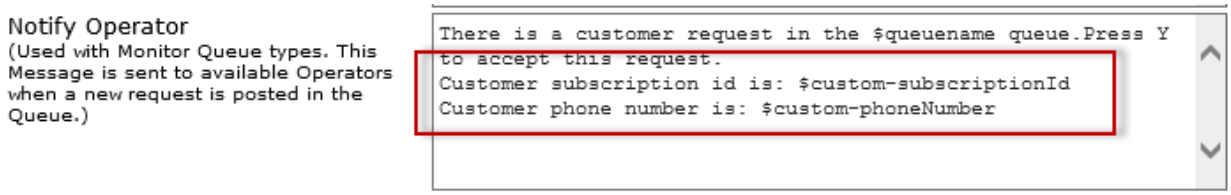
   $custom-<JSON DATA KEY>

   For example if JSON object contained custom property **subscriptionId**

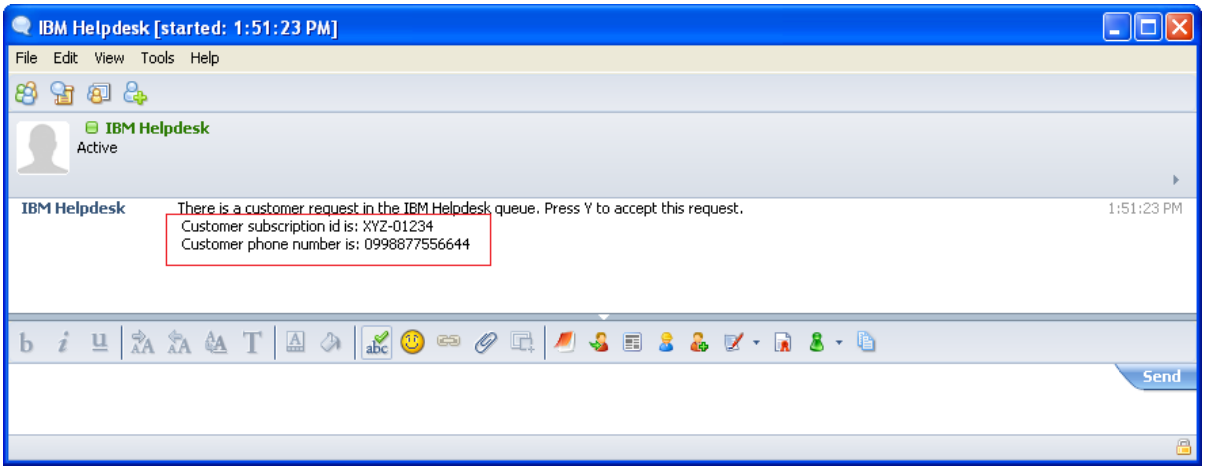
   {"key":"**subscriptionId**","value":"XYZ-1245"}

   then in text message it needs to be accessed as

   $custom-subscriptionId (as in following screenshot)



At runtime it will appear to experts as follows:



Setting up embedded information panel

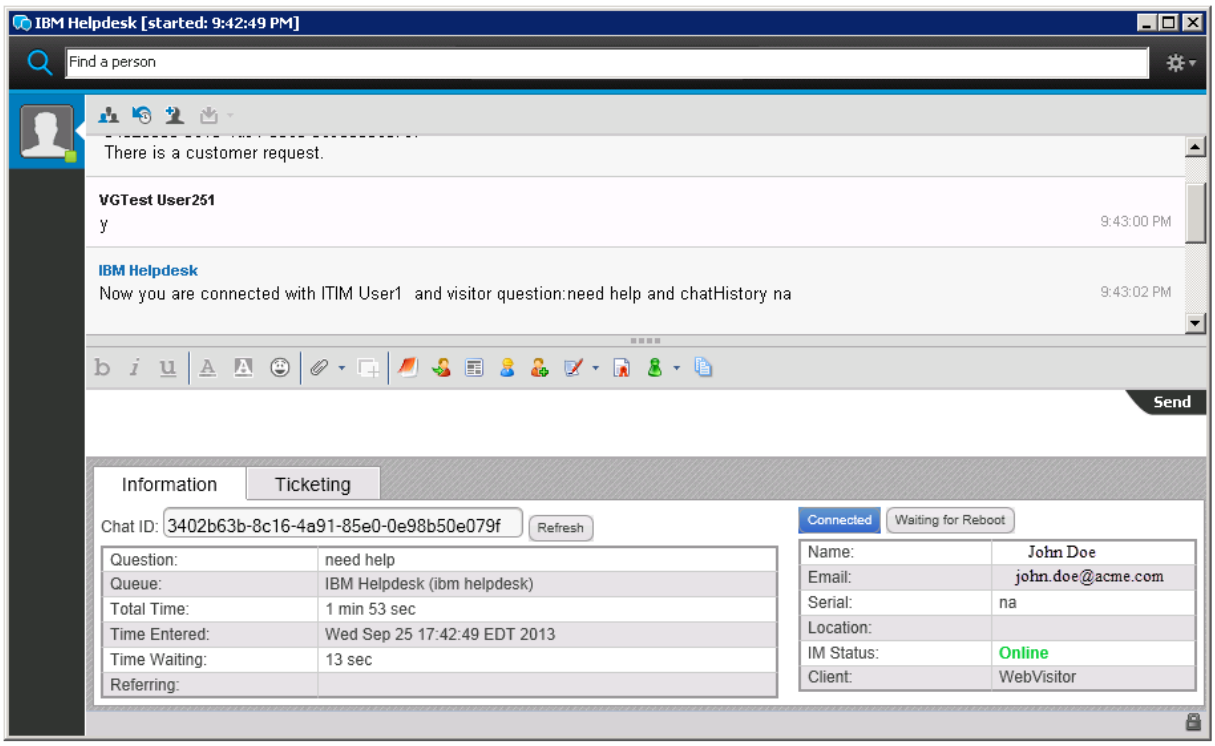
IQMV4 can be configured to render an information panel for experts to display them seeker meta-data/details. These panels are configured and maintained at the queue level and essentially enable a custom HTML (or JSP) page to be attached to the expert's chat session with information on the seeker (customer). This embedded panel is enabled as a feature of the Queue Manager plugin extension for the IBM Sametime client.

In early versions of Queue Manager, the development team developed various extensions and integration options with the IBM Sametime plugin. After numerous builds, and feature requets, it became reasonable to centralize most 'expert level' integration efforts into a simple, and flexible, module that provides 4 main features:

1. When a chat request is accepted, a panel may be attached to the chat session
2. This panel is associated with a the unique ID of the chat request
3. The panel is a standard HTML page that is typically developed using JSP
4. The panel can display information on the seeker and provide other services – such as posting information to a ticketing system

In order to provide the ability to connect the chat session with the embedded HTML panel, the Instant plugin also installs a Javascript to Eclipse 'bridge'. This bridge provides a layer through which information maybe be sent to the panel and the panel, through Javascript, can interact with the chat session.

Information panel looks like the following screen shot:

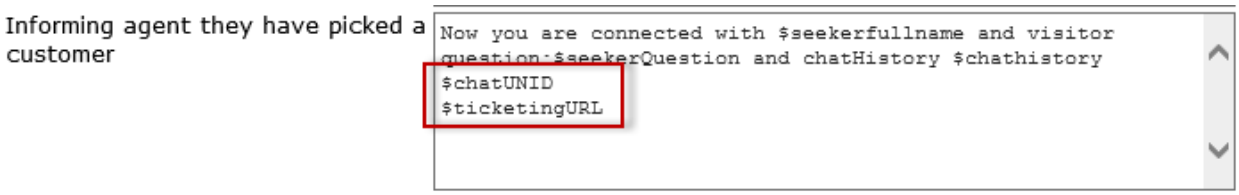


To enable this feature steps are as follows:

1. Edit Queue defintion
2. Go to the Advanced tab
3. Enable CRM option and select the CRM type as “Instant CRM”

4. Next go to “Text” tab under queue properties and edit the message titled “Informing agent they have picked a customer” and add following text variables at the end of the message in a new line:

   $chatUNID
   $ticketingURL



5. Next save the Queue definition.

Experts should now see the information panel when they pick up a seeker request.

Customizing information panel

Layout and data rendered by Information panel can be easily customized. To modify the layout you will need to modify the “SeekerDetails.jsp” (HTML Page) file. It is located at following location (on the system running IQMV 4 engine):

<Apache Install Folder>\Tomcat 6.0\webapps\ITFramework\WEB-INF\jsp\SeekerDetails.jsp

JSP page holds all of the seeker details in MVC model object – “**informationData**”

```
<table id="user">
    <tr class="even">
        <td>Name:</td>
        <td>${informationData.name}</td>
    </tr>
    <tr class="odd">
        <td>Email:</td>
        <td>${informationData.email}</td>
    </tr>
    <tr class="even">
        <td>Serial:</td>
        <td>${informationData.serial}</td>
    </tr>
    <tr class="odd">
        <td>Location:</td>
        <td>${informationData.location}</td>
    </tr>
    <tr class="even">
        <td>IM Status:</td>
        <td><span class="presence">${informationData.imStatus}</span></td>
    </tr>
    <tr class="odd">
        <td>Client:</td>
        <td>${informationData.client}</td>
    </tr>
</table>
```

You can use these details to possibly query your enterprise data sources like Active Directory server/CRM system by placing an AJAX request to fetch and display additional details about the seeker.

Updating information panel to show seeker custom data

Information panel can be customized to show seeker custom meta data. All of the seeker custom properties are contained in informationData MVC model object.

To render a specific custom value in information panel you will need to use following syntax:

${informationData.customDetails.custom<JSON_Data_Key>}

For example if JSON object contained custom property **subscriptionId**

{"key":"**subscriptionId**","value":"XYZ-1245"}

then in Information panel (SeekerDetails.jsp) it needs to be accessed as

${informationData.customDetails.custom-subscriptionId}

HTML code will look like the following screenshot:

```
<tr class="even">
  <td>Name:</td>
  <td>${informationData.hrFirstName}</td>
</tr>
<tr class="odd">
  <td>Email:</td>
  <td>${informationData.mail}</td>
</tr>
<tr class="odd">
  <td>Subscription Id:</td>
  <td>${informationData.customDetails.custom-subscriptionId}</td>
</tr>
```

Note: Prior to the October 2013 release, custom values were referenced in the JSP using the following format:

${informationData.customDetails.<JSON_Data_Key>}

Example here:

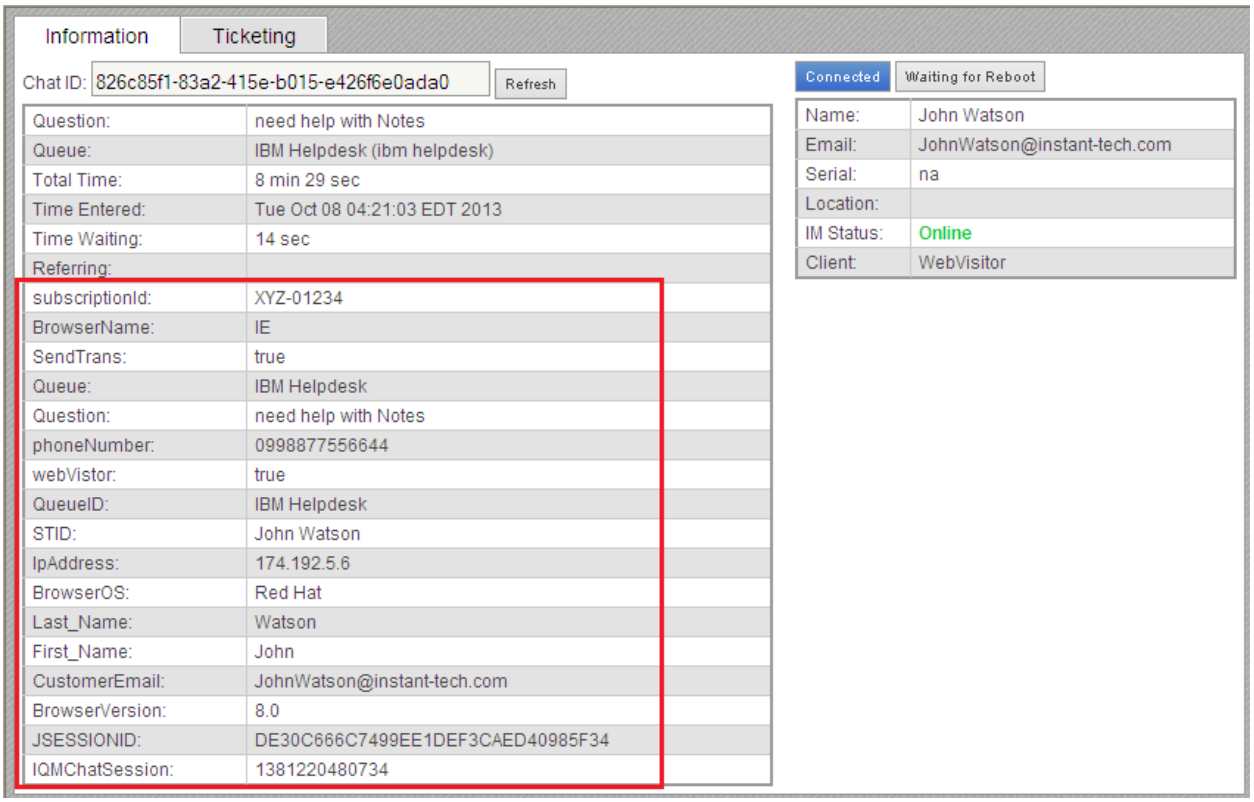
For example if JSON object contained custom property **subscriptionId**

{"key":"**subscriptionId**","value":"XYZ-1245"}

then in Information panel (SeekerDetails.jsp) it needs to be accessed as

${informationData.customDetails.subscriptionId}

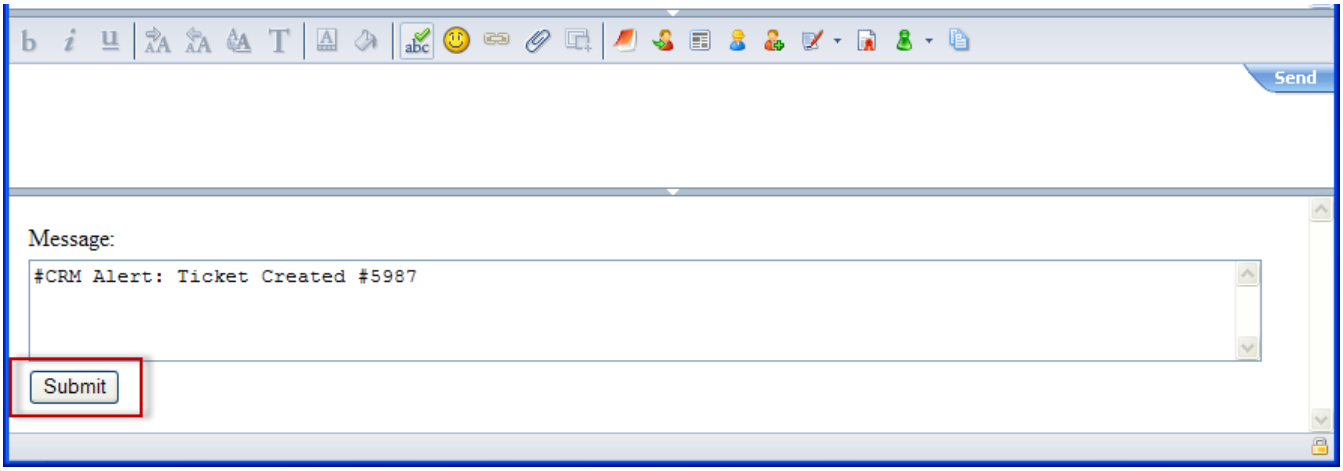
At runtime it will appear to experts as follows:



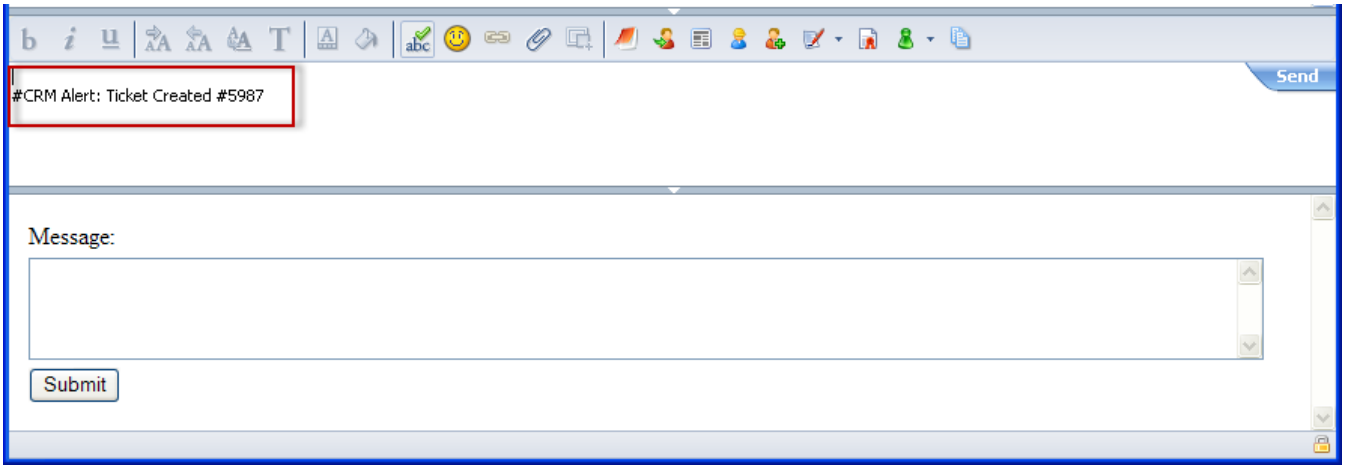
Updating information panel to send out an alert to seeker

IBM Sametime Client exposes **JavaScript Bridge** which allows Information Panel to use the JavaScript callback to post IM messages to seeker.

Message posted using **JavaScript Bridge** will appear in the IM composing text field to the expert (as in following screen shots). Expert can then send it across to seeker.



Text will be posted to IM composing text field using JavaScript callback.



Above example is fairly basic and can be extended for integrating with your existing CRM/Ticketing solution to send out automated notfications to seeker.

JavaScript function call looks like the following snippet:

```
<script language="JavaScript">
        function callJSBridge() {
                var result;
                try {
                        result = JSBridge("<<Alert to send across>>");
                } catch (e) {
                        alert('a java error occurred: ' + e.message);
                        return;
                }
        }
</script>
```

For sample JavaScript Code refer: **JSBridge.html** file located in "**<Tomcat Install Folder>\ webApps\ITFramework\samples\"** folder.

Server Based Extensions

Subscribing to Seeker State events

IQMV4 engine publishes a series of events which can be subscribed by any custom module\bot. You can either write these extensions in **Java** or **Groovy** (http://groovy.codehaus.org). Groovy supports Java coding syntax. In essence it allows you to write/debug/deploy extensions without actually requiring to compile your code.

It allows application developers to integrate Queue workflow with a CRM/Ticketing system.

IQMV4 engine publishes following seeker-state events:

1. **Seeker Arrived**
   This event fires when a seeker places their request in a Queue i.e. when seeker starts a conversation with a Queue.

2. **Seeker Connect**
   This event fires when a seeker gets connected with an expert in a Queue.

3. **Seeker Left**
   This event fires when seeker closes their chat session with the Queue i.e. exits out of the Queue.

To subscribe to Seeker events your class module will need to implement Spring ApplicationListener interface (**org.springframework.context.ApplicationListener**).

Your class should look as follows:

```
public class CaptureSeekerEvents implements ApplicationListener {

      public void onApplicationEvent(ApplicationEvent event) {
            SeekerEvent seekerEvent = (SeekerEvent)event.getSource();

            println("Queue name: " + seekerEvent.getQueueName());
            println("Seeker ID: " + seekerEvent.getSeekerID());

            if (seekerEvent.getEventType().equalsIgnoreCase("ARRIVED") {
                  //Seeker Arrived
            }

            if (seekerEvent.getEventType().equalsIgnoreCase("CONNECTED") {
                  //Seeker Connected
                  println("Expert ID: " + seekerEvent.getExpertID());
            }

            if (seekerEvent.getEventType().equalsIgnoreCase("LEFT") {
                  //Seeker Left
            }
      }
}
```

For a complete sample refer CMS.groovy file located at "<Tomcat Install Folder>\webApps\ITFramework\WEB-INF\classes\groovyBots"

Hooking into IQMV4 engine events and collections

IQMV4 is a Spring based J2EE application. Application is a collection of beans (modules) and data collections. You can connect to these beans and collections to:

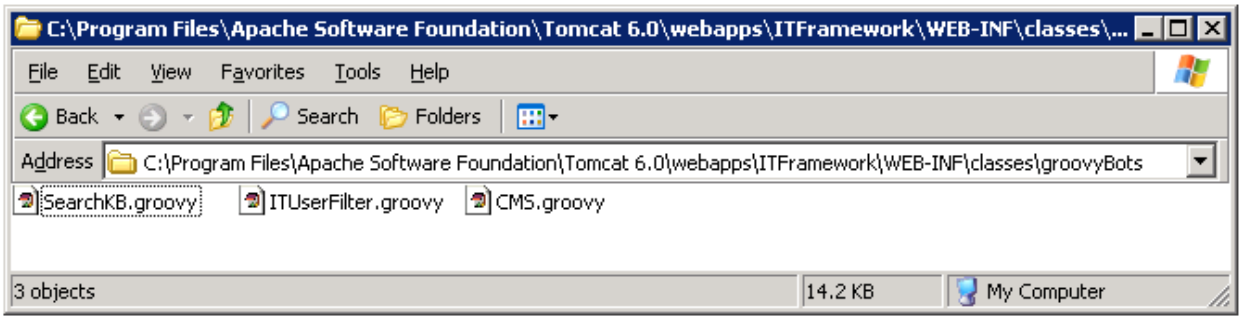
1. Get a list Queues\Interviews that are currently running
2. Get notified when a Queue or Interview is started\stopped
3. Get a list of waiting seekers in Queue
4. Get Seeker properties
5. Get Queue\Interview properties
6. Get a list of experts
7. Get expert state in a Queue
8. Get access to chat logging service
9. Get access to email service

While it is recommended to access these details using provided web service calls there might be instances where you would want to access these beans and collections directly to get real time notifications. To get more details about natively hooking into the IQMV4 engine please contact us at support@instant-tech.com

Deploying Groovy Extensions

Take following steps to deploy your extensions:

1. All of the **extensions** should be placed in following folder on server: "**<Tomcat Install Folder>\webApps\ITFramework\WEB-INF\classes\groovyBots**"



2. After placing your extension into groovyBots folder next you will need to register them in "**InterviewContext_groovy.xml**" file located at "**<Tomcat Install Folder>\ webApps\ ITFramework\WEB-INF**"

File looks like the following screen shot:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:lang="http://www.springframework.org/schema/lang"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/lang
    http://www.springframework.org/schema/lang/spring-lang-3.2.xsd">
    <lang:groovy id="CMS" refresh-check-delay="5" script-source="classpath:groovyBots/CMS.groovy" ></lang:groovy>
    <lang:groovy id="ITUserFilter" refresh-check-delay="5" script-source="classpath:groovyBots/ITUserFilter.groovy" ></lang:groovy>
</beans>
```

3. Add the following text to register your extension:
   <lang:groovy id="**<Extension_Unique_ID>**" refresh-check-delay="5" script-source = "classpath:groovyBots/**<Your_Extension_FileName.groovy>**" > </lang:groovy>

4. After updating "**InterviewContext_groovy.xml**" file you will need to restart Apache Tomcat Service to load/run your extension.

Sending Email Alerts

IQMV4 engine has a built-in email module which can be accessed by any custom module\bot.

To access the email module you will need to use following code snippet:

```
//Load mail module
ITEmail mailModule = (ITEmail) new RuntimeManager().getBotObject("ITEmail");

//Specify mail properties
mailModule.setHost(<SMTP_SERVER_FQDN>);
mailModule.setPort(<SMTP_SERVER_PORT>);
mailModule.setFrom(<Sender_Email_Address>);
mailModule.setPassword(<SMTP_Account_Password>);

mailModule.setTo(<Recipient_Email_Address>);
mailModule.setSubject(<Email_Subject>);
mailModule.setMessage(<Email_Message>);

//dispatch email
mailModule.sendEmail();
```

For a complete sample refer CMS.groovy file located at "<Tomcat Install Folder>\webApps\ITFramework\WEB-INF\classes\groovyBots"

Initiating outbound chat request – to contact a seeker

Instant Queue Manager exposes an API call which allows you to place outbound chat request in Queue. Queue verifies if the seeker is available, subsequently reaches out to them and automatically places their request in Queue.

To place Outbound chat request you will need to submit request as JSON object. Structure of JSON object should be as follows:

{"CRM":[
{"key":"webVistor","value":"false"},
{"key":"**STID**","value":"John Doe"},
{"key":"SametimeName","value":"John Doe"},
{"key":"First_Name","value":"John"},
{"key":"Last_Name","value":"Doe"},
{"key":"**Question**","value":"Reset my password"},
{"key":"OutBoundRequest","value":"true"},
{"key":"**QueueID**","value":"**Instant Sales**"}]}

Request should contain seeker Sametime ID, their initial query and Queue login name.

JSON data needs to be sent to following URL:
http://<tomcatServerFQDN>/ITFramework/servlet/CRMServlet?datajson=<JSON_Request>

Server will return back server response as:

SUCCESS (if data was successfully posted)

Or

ERROR (if JSON is malformed)

For sample JavaScript Code refer: **outbound.html** file located in "**<Tomcat Install Folder>\ webApps\ITFramework\samples\"** folder.

Sample demonstrates process of submitting Outbound chat request.