



Intelligent Search Application Development Guide

A Guide to Customizing and Extending InQira

InQira Version 8.2
Document Number IS82-API-00
March 3, 2010

InQira, Inc.
900 Cherry Ave., 6th Floor
San Bruno, CA 94066

COPYRIGHT INFORMATION

Copyright © 2002 - 2010 InQuira, Inc.
Product Documentation Copyright © 2003 - 2010 InQuira, Inc.

RESTRICTED RIGHTS

This document is incorporated by reference into the applicable license agreement between your organization and InQuira, Inc. This software and documentation is subject to and made available only pursuant to the terms of such license agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy, modify, disassemble or reverse engineer the software and documentation, except as specifically allowed in the license agreement and InQuira will take all necessary steps to protect its interests in the software and documentation. To the extent certain third party programs may be embedded into the InQuira software, you agree that the licensors for such third party programs retain all ownership and intellectual property rights to such programs, such third party programs may only be used in conjunction with the InQuira software, and such third party licensors shall be third party beneficiaries under the applicable license agreement in connection with your use of such third party programs.

This document may not, in whole or in part, be photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without written prior consent from InQuira, Inc., which may be withheld in its sole and absolute discretion.

The information in this document is subject to change without notice and does not represent a commitment on the part of InQuira, Inc. The documentation is provided "AS IS" without warranty of any kind including without limitation, any warranty of merchantability or fitness for a particular purpose. Further, InQuira, Inc. does not warrant, guarantee, or make any representations regarding the use, or the results thereof. Although reasonable measures have been taken to ensure validity, the information in this document is not guaranteed to be accurate or error free.

TRADEMARKS AND SERVICE MARKS

InQuira, Inc., InQuira 8, InQuira 7, InQuira 6, InQuira 5, InQuira Natural Interaction Engine, Information Manager, Call Center Advisor, and iConnect are trademarks or registered trademarks of InQuira, Inc.
Sentry Spelling-Checker Engine Copyright © 2000 Wintertree Software, Inc.

All other trademarks and registered trademarks contained herein are the property of their respective owners.

Contents

Preface: About This Guide	1
In This Guide	2
Contacting InQuira	3
InQuira Product Documentation	4
Intelligent Search Documentation	4
InQuira Analytics Documentation	6
Information Manager Documentation	6
Contact Center Documentation	7
Screen and Text Representations	7
References to World Wide Web Resources	7
Chapter 1 The InQuira User Interface	9
The Personalized Response User Interface	9
User Interface Processing	10
Application Response Format	10
The Parameters Section	10
The Answers Section	11
The Query Section	12
Chapter 2 User Interface Components	13
The Main Template	13
Main Template File Example	14
The Global Layout Style Templates	15
Basic Search Layout Display Example	16
Request and Response Element Templates	17
Global Configuration Parameters Template	17

Sample Configuration Parameters File	17
Request Element Templates	18
Request Area Example	19
Dialog Request Area Example	20
Response Element Templates	20
Global Elements and Utilities	21
Chapter 3 User Interface Elements	23
Request Elements	23
Response Elements	24
Answer Display Features	25
Answer Purposes	26
Default Answer Purposes	27
Answer Portlets	28
Default Answer Portlets	29
Promotions Portlet Example	30
Act Now Portlet Example	31
Learn More Portlet Example	32
Definition Portlet Example	33
Feature Content Portlet Example	34
Chapter 4 Customizing the User Interface	35
Specifying the User Interface Layout	35
Integrating the User Interface	36
Customizing Style Elements	36
Customizing General Style Elements	36
Customizing Question Area Definitions	38
Customizing Answer Area Definitions	39
Customizing Sidebar Area Definitions	41
Customizing Request Elements	43
Customizing the Request Heading	44
Customizing the Example Question	44
Customizing the Question Box	45
Customizing the Tips Link	45
Customizing the Submit Button	45
Customizing Response Elements	46
Customizing the Question Echo	46
Customizing the Answer Introduction	47
Customizing Answer Headings	47
Customizing the Answer Body Text	47
Customizing the Answer Document Link	48
Configuring Answer Purposes	48

Adding Answer Purposes to the Application	49
Customizing Answer Portlets	52
Specifying Portlet Display Position	52
Customizing Portlet Headings	52
Customizing Portlet Answer Headings	53
Customizing Portlet Answer Text	53
Customizing Portlet Document Links	53
Chapter 5 Implementing Optional Features	55
The Process Wizard User Interface	56
The Process Wizard Answer	56
The Step Display Area	57
Modifying the Process Wizard User Interface	57
Activating the Personalized Navigation User Interface Layout	59
The Personalized Navigation User Interface Elements	59
Personalized Navigation XSL Style Sheet Elements	60
Personalized Navigation CSS Style Sheet Elements	60
Personalized Navigation-Related XML Elements	61
Implementing Direct Page Display	61
Direct Page Display Example	62
Implementing a Virtual Representative	63
Implementing User Feedback Collection	63
The User Feedback Portlet	64
The User Feedback Comment Form	65
The User Feedback Process	66
Customizing the User Feedback Area Heading	67
Customizing the User Feedback Rating Labels	68
Customizing the User Feedback Comment Form	69
Disabling the User Feedback Feature	69
Implementing Click-Through Logging	70
Highlighting Answers Within Documents	71
Enabling Highlighting within Answer Documents	72
Specifying HTML Highlighting Style Attributes	73
Managing Multiple Languages in the User Interface	75
Chapter 6 Creating a Custom Content Crawler	77
Example: Creating a Database Web Crawler	77
Example: Configuring the Database Web Crawler	81
Configuring a Custom Crawler	83
Example Crawler Settings	84

Chapter 7	Creating a Custom Document Preprocessor	85
	Example: Creating a Document Preprocessor	85
	Configuring a Custom Document Preprocessor	89
	Supporting Multiple Navigation Applications	90
Chapter 8	Creating a Custom Task	91
	Example: Creating a Simple Custom Task	92
	Example: Handling Argument Parsing	94
	Example: Handling Document Count and Progress Updates	98
	Example: Handling User Task Interruptions	101
	Configuring a Custom Task	103
Chapter 9	Creating a Custom Authentication Interface.....	105
	Example: Creating a Simple Custom Authenticator	106
	Example: Simple Unit Testing of a Custom Authenticator	108
	Example: Configuration-based Test for IAuthenticator Objects	110
	Configuring a Custom Authenticator	111
Chapter 10	Integrating an External Authentication Application.....	113
	Example: Integrating a Delegation Authenticator	114
	Example: Integrating a Delegation Detector	116
	Configuring a Delegation Authenticator or Detector	118
Chapter 11	Creating an Action Plugin.....	119
	Example: Creating an Action Plugin	119
	Configuring an Action Plugin	121
Chapter 12	Creating a Custom Preference Handler	123
	Example: Creating a Preference Handler	123
	Configuring a Preference Handler	124
Chapter 13	Rendering Web Pages Using a Custom Agent.....	125
	Example: Rendering a Web Page Using a Custom Agent	125

This guide provides information about integrating and customizing the InQuira 8.1 Personalized Response User Interface. It describes the components and elements that make up the User Interface, and includes guidelines for:

- Incorporating the User Interface into your web architecture
- Customizing User Interface appearance and functionality
- Implementing special features

This guide also provides information for application developers who want to customize or extend InQuira 8.1 through its API. For information on configuration-based changes or changes to InQuira 8.1 that are not accomplished through its API, refer to the *Intelligent Search Optimization Guide*.

For a full discussion of the InQuira 8.1 architecture, components, and instances refer to the *Intelligent Search Administration Guide*. For information about setting up the development environment and deploying customizations and code changes to the production environment also refer to the *Intelligent Search Administration Guide*.

This preface includes information on:

- The general organization of this guide
- The support services available from InQuira Customer Support
- The available product documentation

In This Guide

The Intelligent Search Application Developer's Guide is divided into the following sections:

<i>Chapter 1, The InQuira User Interface</i>	This section describes the basic functions of the User Interface and input format of the application responses.
<i>Chapter 2, User Interface Components</i>	This section describes the templates that define the User Interface functionality and presentation.
<i>Chapter 3, User Interface Elements</i>	This section describes the various request and response elements within the User Interface.
<i>Chapter 4, Customizing the User Interface</i>	This section describes the process of specifying User Interface layout, individual element styles, and implementing Personalized Response features.
<i>Chapter 5, Implementing Optional Features</i>	This section describes optional User Interface features that you can use within your application.
<i>Chapter 6, Creating a Custom Content Crawler</i>	This section shows you how to implement a custom DB Web crawler.
<i>Chapter 7, Creating a Custom Document Preprocessor</i>	This section discusses common preprocessing tasks and provides an example based on which you can develop your own preprocessing routines.
<i>Chapter 8, Creating a Custom Task</i>	This section shows you how to create a custom task.
<i>Chapter 9, Creating a Custom Authentication Interface</i>	This section shows you how to create a custom authentication interface.
<i>Chapter 10, Integrating an External Authentication Application</i>	This section shows you how to integrate InQuira 8.1's authentication with an external application.
<i>Chapter 11, Creating an Action Plugin</i>	This section shows you how to create and integrate an action plugin that executes when a rule is invoked.
<i>Chapter 12, Creating a Custom Preference Handler</i>	This section provides a template for developing preference handlers.

*Chapter 13, Rendering
Web Pages Using a
Custom Agent*

This section provides an example of how to integrate a custom agent bypassing the web page rendering functionality built into InQuira 8.1.

Contacting InQuira

You can contact InQuira by mail, telephone, fax, and email.

Address:	InQuira, Inc. 900 Cherry Ave., 6th Floor San Bruno, CA 94066
Telephone:	(650) 246-5000
Fax:	(650) 264-5036
Email:	For sales information, send email to sales@inquira.com . For product support, send email to support@inquira.com .
World Wide Web:	Learn more about InQuira products, solutions, services, and support on the world wide web at: www.inquira.com .

InQuira Product Documentation

InQuira documentation is available only to licensed users of our software products and may not be redistributed in any form without express permission from InQuira, Inc.

The InQuira documentation is available in PDF format. Customers can download the PDF files from:

<http://documentation.inquira.com/>

NOTE: You need a PDF reader application installed on each processor on which you plan to view the InQuira product documentation. The Adobe Acrobat reader is available from Adobe Systems at: <http://www.adobe.com>.

Detailed information about each product document set is available in:

- [Intelligent Search Documentation on page 4](#)
- [InQuira Analytics Documentation on page 6](#)
- [Information Manager Documentation on page 6](#)
- [Contact Center Documentation on page 7](#)

If you encounter a problem, need help using the documentation, or want to report an error in the content, please contact InQuira Customer Support.

If you need help obtaining InQuira product documentation, or want to obtain permission to redistribute a portion of the contents, please contact your InQuira account representative.

Intelligent Search Documentation

Intelligent Search is distributed with the following documentation.

Document	Number	Description
Intelligent Search Installation Guide	IS80-IG-00	This guide is intended for technical staff who are responsible for installing InQuira 8.1. It provides detailed information on installing InQuira 8.1 and configuring the application on a single processor using the Installation Configuration Environment facility.
Intelligent Search Administration Guide	IS80-CA-00	This guide is intended for system and application administrators who need to configure an InQuira 8.1 application in an enterprise environment. It describes InQuira 8.1 integration, development, configuration, and maintenance processes and tasks.

Intelligent Search Language Administration Guide	IS80-LA-00	This guide is intended for business users and subject matter experts who need to create and maintain the language processing elements of an InQuira 8.1 application using the System Manager. This book provides usage information about the System Manager, conceptual information about the InQuira 8.1 language objects, and task information about the process of managing the user experience provided by the InQuira 8.1 application.
Intelligent Search Language Tuning Guide	IS80-LD-00	This guide is intended for application developers who need to create and maintain advanced InQuira 8.1 language-processing elements using the Dictionary and other InQuira Language Workbench applications.
Intelligent Search Optimization Guide	IS80-AG-00	This guide is intended for application developers who need to implement InQuira 8.1 advanced features, including Personalized Navigation and Process Wizards.
Intelligent Search Application Development Guide	IS80-API-00	This guide provides information about integrating and customizing the InQuira 8.1 Personalized Response User Interface.
Intelligent Search Language Reference	IS80-LRG-00	This guide is for language developers implementing InQuira 8.1 applications that utilize the intent libraries and advanced language processing functions. These guides are published as separate documents that provide reference information for each industry-specific intent library. Each reference also contains complete descriptions of InQuira Match Language and Variable Instantiation Language.
Intelligent Search User Interface Guide	IS80-UI-00	This guide is intended for application developers who need to customize the InQuira 8.1 Personalized Response User Interface, and integrate it with a production web application. It contains information about the elements and features of the User Interface, and provides guidelines for integrating it into an enterprise web architecture, customizing its appearance and functionality, and implementing various special features.

InQuira Analytics Documentation

InQuira Analytics is distributed with the following documentation.

Document	Number	Description
InQuira Analytics Installation Guide	IA80-IG-00	This guide is intended for technical staff who are responsible for installing InQuira Analytics. It provides detailed information on installing and configuring the InQuira Analytics product for use with an InQuira 8.1 application.
Analytics User Guide	IA80-CA-00	This guide is intended for systems and application administrators who need to configure the Intelligent Search and Information Manager Analytics components to report on InQuira 8.1 application performance.

Information Manager Documentation

InQuira Information Manager is distributed with the following documentation.

Document	Number	Description
Information Manager Installation Guide	IM80-IG-00	This guide is intended for technical staff who are responsible for installing InQuira Information Manager. It provides detailed information on installing and configuring the Information Manager product.
Information Manager Administration Guide	IM80-CA-00	This guide is intended for systems and application administrators who need to configure and administer an InQuira Information Manager application, and integrate it with an InQuira 8.1 application. It also contains information for general business users who need to use the Information Manager to create and manage content.
Information Manager Content Authoring Guide	IM80-AG-00	This guide is intended for technical staff who are responsible for authoring content in InQuira Information Manager. It provides detailed information on creating content and managing workflow tasks in the Information Manager console.
Information Manager Developer's Guide	IM80-WSR-00	This guide is intended for application developers who need to integrate Information Manager content, content category, and user and security functions with external applications. It contains reference information and examples for all packages, classes, methods, and interfaces of the Information Manager Web Services API.

Contact Center Documentation

The InQuira 8.1 contact center products are distributed with the following documentation.

Document	Number	Description
Contact Center Advisor Integration Guide	CA80-IG-00	This guide is intended for application developers and systems administrators who need to plan for and integrate the InQuira Contact Center Advisor with an InQuira application and a supported CRM application.
Intelligent Search Siebel Integration Guide	CAS80-IG-00	This guide is intended for application developers and systems administrators who need to plan for and integrate InQuira 8.1 with Siebel 7 Enterprise Applications using the Siebel Adapter for InQuira 8.1.

Screen and Text Representations

The product screens, screen text, and file contents depicted in the documentation are examples. We attempt to convey the product's appearance and functionality as accurately as possible; however, the actual product contents and displays may differ from the published examples.

References to World Wide Web Resources

For your convenience, we refer to Uniform Resource Locators (URLs) for resources published on the World Wide Web when appropriate. We attempt to provide accurate information; however, these resources are controlled by their respective owners and are therefore subject to change at any time.

Chapter 1 The InQuira User Interface

The InQuira 8.1 Personalized Response User Interface is a full-featured graphical user interface designed to integrate easily with your existing production web site. The User Interface provides the elements required for processing requests and presenting responses, and supports additional optional features that you can implement as desired.

To use the User Interface in a production web environment, you must:

- Integrate it into your web site's navigation and presentation scheme
- Customize it to conform to your organization's functional and presentation requirements
- Implement any desired optional features as described in [Chapter 5, Implementing Optional Features](#)

The User Interface is installed as part of the standard product installation.

NOTE: The User Interface is available only as an HTML-based user interface for use with a configured InQuira 8.1 web application. For information about implementing InQuira 8.1 using other technologies, contact your InQuira account representative.

The Personalized Response User Interface

The InQuira 8.1 User Interface incorporates InQuira's Personalized Response concept, which presents direct answers to user requests in its main answer area, and categorized related information in that you configure within the Dictionary.

The Personalized Response User Interface organizes various types of related responses into separate graphical areas, or portals, enabling you to establish consistent, focused, and targeted presentation for various types of application content, such as general site information, online glossaries, promotional material, and site features, such as calculators and other tools.

User Interface Processing

The User Interface contains all of the elements required to solicit user questions and present categorized application responses. During request processing, the User Interface:

- Passes user input to the application for processing. See [Chapter 1, Dictionary Manager Advanced Features](#) in the *Intelligent Search Optimization Guide* for an overview of application request and response processing.
- Receives formatted responses from the application. See [Application Response Format on page 10](#) for information about the response format.
- performs final formatting and displays responses to the end user, as specified by the configured presentation elements as described in [Chapter 4, Customizing the User Interface](#).

Application Response Format

The application passes responses to the User Interface as a file that conforms to an internal Extensible Markup Language (XML) document type definition (DTD). The User Interface templates are stylesheets that transform the XML into formatted HTML for presentation within a browser.

The response file is divided into sections:

- [The Parameters Section on page 10](#)
- [The Answers Section on page 11](#)
- [The Query Section on page 12](#)

The Parameters Section

The parameters section provides meta-information about the response, such as context information and other configuration parameters. The User Interface uses this information to retrieve page parameters, server URLs, and other required information.

The following example is an excerpt from a typical parameters section.

```
<params>
<param name="type">AnswerQuestion</param>
<param name="Question">how much can I contribute to a Roth ira in?
<param name="baseURL">http://lcdemo2:8222/htmlagent/ui.jsp</param>
</params>
```

The Answers Section

The answers section contains the various content responses (answers) to the request (question). Factors that determine the number of answers passed to the User Interface include:

- The number of content matches (answers) located in the application content
- The scores associated with the located answers

The results file groups answers according to answer purposes, which are specified in the Dictionary. The User Interface displays answers associated with each purpose in a specific section, or portlet of the response page. The maximum number of answers within each portlet is determined by display thresholds. See [Configuring Answer Purposes on page 48](#) for more information about answer purposes and how they are displayed by the User Interface.

The following example includes a general answer and an answer assigned to the purpose `link to category`.

```
<answer score="1.0">
  <answer type="unstructured" score="0.6691748880962431"
  <section>
    <title idx="0"
    <snippet lvl="0">Financial
    </title>
    <text idx="1"
    <snippet lvl="1">Only married couples with
    <snippet lvl="3"> $ 150 </snippet>
    <snippet lvl="1"> , 000 or less and singles
    .
    .
    .
    </text>
  </section>
  <highlighted_link
  <similar_response_link
  </answer>
</answer>
<link_to_category score="1.0">
<answer type="custom" score="1.0">
  <sentence type="code">&lt;a
  <title type="code">Roth IRA</title>
</answer>
```

The Query Section

The query section contains history information associated with the session, such as previously asked questions. The User Interface uses this information to present session information with results.

The following example is an excerpt from a typical query section.

```
<query>
<question transactionId="1">
<original>how much can I contribute to a Roth ira in
<paraphrase>how much can I contribute to a Roth ira in
</question>
</query>
```

Chapter 2 User Interface Components

The User Interface consists of a set of templates that use Extensible Stylesheet Language Transformation (XSLT) and HTML Cascading Style Sheets (CSS) to define presentation characteristics.

The set of templates includes the main template, called `main.xsl`, and subordinate templates that contain the elements required for User Interface implementation.

The templates contain presentation and navigation design elements, such as:

- Page layouts
- Functional elements, such as user input elements and response presentation elements
- Global elements, such as color schemes and font families

The templates are pre-configured with default values for required elements.

In addition to the required User Interface elements, the templates contain elements that support optional features, such as Personalized Navigation, direct page display, and dialog-style user interaction.

See [Chapter 3, *User Interface Elements*](#) for more information about the elements of the User Interface.

The Main Template

The main template specifies the set of subordinate templates that determine the layout, functional elements, and style of the User Interface. The main template also specifies additional utilities and directories that provide basic functional or graphical elements.

You need to modify the main template to integrate the User Interface with your site's navigation structure. The main template is located in:

```
<InQuira_home>/inquire/int/xsl/search
```

The main template specifies subordinate templates as include statements. [Main Template File Example on page 14](#) contains a sample section of the main template showing its structure.

See [Chapter 4, Customizing the User Interface](#) for more information on using the main template.

Main Template File Example

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<!-- General -->
<xsl:import href="config.xml"/>
<xsl:import href="globals.xml"/>
<xsl:import href="includes.xml"/>
<xsl:import href="../common/util.xml"/>
<!-- Options for Search UI Main Screens -->
<xsl:import href="ui_search_basic.xml"/>
<!--xsl:import href="ui_search_and_browse.xml"/-->
<!--xsl:import href="ui_search_vrep.xml"/-->
<!-- Other search UI pages -->
<xsl:import href="instant_answer_page.xml"/>
<xsl:import href="user_comments_page.xml"/>
<!-- Search UI Main Areas -->
<xsl:import href="results.xml"/>
<xsl:import href="sidebar.xml"/>
<xsl:import href="structured_details.xml"/>
<xsl:import href="tips.xml"/>
<xsl:import href="error.xml"/>
<!-- End of Imports -->
<xsl:output method="html" indent="yes"/>
<xsl:strip-space elements="*" />

<!-- Override the default, empty resource file with our own for the search UI -->
<xsl:variable name="resource-file" select="document('resource.xml')" />

<xsl:template match="/">
  <xsl:choose>
    <xsl:when test="$error-message">
      <xsl:call-template name="error-page" />
    </xsl:when>
    <xsl:when test="$show-user-comments-page">
      <xsl:call-template name="user-comments-page" />
    </xsl:when>
    <xsl:when test="$show-instant-answer-frame">
      <xsl:call-template name="instant-answer-frame" />
    </xsl:when>
    <xsl:when test="$instant-answer and not($no-jump or $show-definition-detail-page or
```

```

$show-structured-detail-page)">
  <xsl:call-template name="instant-answer-page" />
</xsl:when>
<xsl:otherwise>
  <xsl:call-template name="question-and-results-page" />
</xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>

```

The Global Layout Style Templates

The layout style templates determine the basic format of the User Interface request and response pages. You specify the following layout templates using an include statement in the main template file.

- The standard response page format (`ui_search_basic.xml`)
- The Personalized Navigation layout (`ui_search_and_browse.xml`)
- The Virtual Representative layout for (`ui_search_vrep.xml`)

The standard response page template is enabled by default, as shown in the following example:

```

<!-- Options for Search UI Main Screens -->
<xsl:import href="ui_search_basic.xml"/>
<!--xsl:import href="ui_search_and_browse.xml"/-->
<!--xsl:import href="ui_search_vrep.xml"/-->

```

See [Chapter 5, Implementing Optional Features](#) for more information on enabling the alternate global layout styles.

Basic Search Layout Display Example

The basic search layout provides a large left-column answer area, and the question input area and related information portlets arrayed in the right column. [Answer Display Features on page 25](#) describes features of the answer displays.

The screenshot shows a search results page for 'How much can I contribute to an IRA?'. The page is divided into a main content area on the left and a right-hand sidebar with several portlets.

Search Results:

ANSWERS

LM Finance | IRA Contribution - Annual
Maximum IRA contributions are determined by the tax year and your age during the tax year. You can contribute up to the following amounts to a Traditional or Roth IRA:

Tax Year	Under Age 50	Age 50 and Older
2001	\$2000	\$2000
2002-2004	\$3000	\$3500
2005	\$4000	\$4500
2006-2007	\$4000	\$5000
2008 and Later	\$5000	\$6000

LM Finance | Traditional IRA
Traditional Individual Retirement Accounts (IRAs) are available to employed persons and their spouses. The amount that you can contribute to an IRA is limited, and the limits are determined by the date and your age. IRA contributions are completely deductible for taxpayers and spouses who do not participate in a qualified plan. For plan participants, contributions are deductible only if their adjusted gross income is below a defined limit. IRA earnings are tax-deferred, meaning that they are not taxed until withdrawn.

LM Finance | IRA Contributions
\$3,000 per individual taxpayer or non-working spouse (Spousal IRA) or up to 100% of earned income (if less than maximum), excluding any contributions to a Roth IRA. Add \$500 for contributors over age 50 in contribution year.
[1 Similar Answer](#)

LM Finance | IRA FAQ
A Traditional IRA is a special retirement account that you can start at any time to save for retirement. Traditional IRAs are also called contributory IRAs. You can make annual contributions to an IRA up to \$3,000 or 100% of your earned income, whichever is less. You can also contribute the same amount to a separate Spousal IRA if your spouse does not work. Contributions to IRAs are tax deductible, within specified limits, and their earnings are tax-deferred (not taxed) until withdrawal at eligibility (age 70 ½).

LM Finance | Roth IRA Comparison
Provides for tax-exempt income at eligibility date. Contribute after-tax earnings. Can supplement employer's plan. Can contribute to Roth IRA and Traditional IRA in the same year (up to \$3,000 or \$3,500 age 50 and over).

LM Finance | Traditional IRA Comparison
Provides tax-deferred contributions. Income taxed at retirement. Can supplement employer's plan. Can contribute to Roth IRA and Traditional IRA in the same year (up to \$3,000 or \$3,500 age 50 and over).

Right-hand Portlets:

- Have Another Question?** Type it below to find an answer now. Includes a search input field, a 'Tips' link, and an 'Ask' button.
- Act Now**
 - [Plan My Retirement Savings](#)
 - [Am I Eligible?](#)
 - [Open My IRA Today](#)
- Related Links**
 - [Retirement Accounts](#)
 - [IRA FAQ](#)
 - [Retirement Services](#)
- Definitions**
 - [IRA](#)
 - [IRA - Spouse](#)
 - [IRA - Education](#)
 - [IRA - Roth](#)
- Are we answering your questions?**
 - Absolutely!
 - Usually
 - Sometimes
 - Rarely
 - Not even close!

Submit

Request and Response Element Templates

The request and response element templates determine the basic format and content of the request and response elements within the specified layout.

- [Sample Configuration Parameters File on page 17](#)
- [Request Element Templates on page 18](#)
- [Response Element Templates on page 20](#)

Global Configuration Parameters Template

The configuration parameters template specifies global settings for both request and response elements. The `config.xsl` template contains User Interface configuration parameters, such as section headers and feature switches. [Sample Configuration Parameters File on page 17](#) provides a sample of the file contents.

Sample Configuration Parameters File

The following is a sample of the configuration parameters file, `config.xsl`.

```
<?xml version="1.0" ?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<!--
- Configurable variables
-->

<!--
- Score Thresholds
-
- best-answers-min-score: Minimum score required to be considered one of the best answers
- best-answers-min-diff: Minimum difference between scores required before being cut off from
the best answers
- best-answers-max-display: Maximum # of best answers to display
-->
<xsl:variable name="best-answers-min-score"      select="0.90" />
<xsl:variable name="best-answers-min-diff"      select="0.01" />
<xsl:variable name="best-answers-max-display"   select="3" />

<!--
- The spellchecker returns suggestions with scores between 0 and 100.
- This sets the minimum score required before a suggestion is made to the user.
-->
```

```

<xsl:variable name="spellcheck-min-suggest-score" select="90" />
<!--
- User Interface Options
-->
<xsl:variable name="default-charset"          select="UTF-8" />
<xsl:variable name="get-user-feedback"        select="true()" />
<xsl:variable name="show-search-running-indicator" select="true()" />
<xsl:variable name="debug-full-excerpt"      select="false()" />

</xsl:stylesheet>

```

Request Element Templates

The request element templates determine the basic format and contents of the request elements within a specified layout.

Request elements

`question.xml`

This template specifies standard question interaction using the question boxes, example questions, and other user input elements. You must specify this template or the alternative dialog-style elements. This template is the default. [Request Area Example on page 19](#) provides a sample request area display.

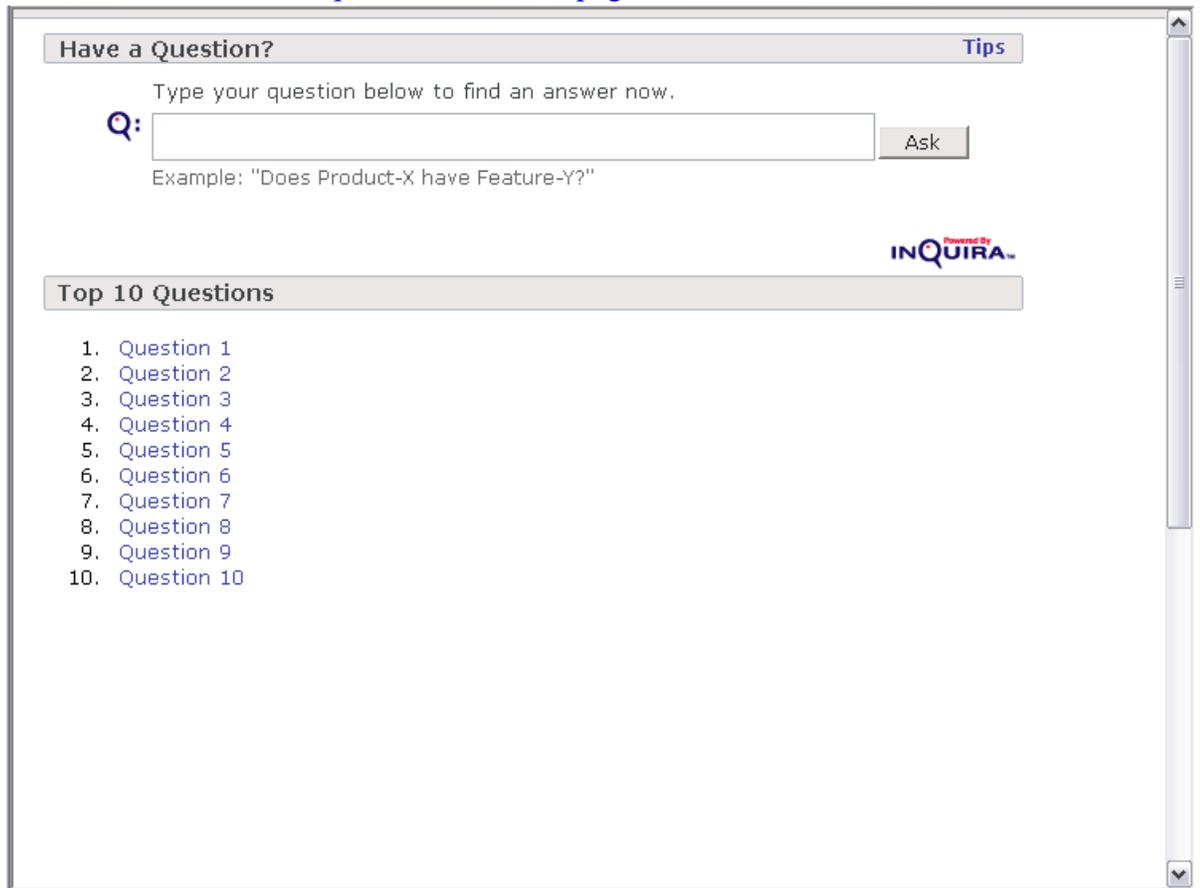
Dialog-style elements

`question_vrep.xml`

This template specifies dialog-style question interaction for use with virtual representatives (VREPs) or similar implementations, as described in [Implementing a Virtual Representative](#). [Dialog Request Area Example on page 20](#) provides a sample dialog-style request area display.

Request Area Example

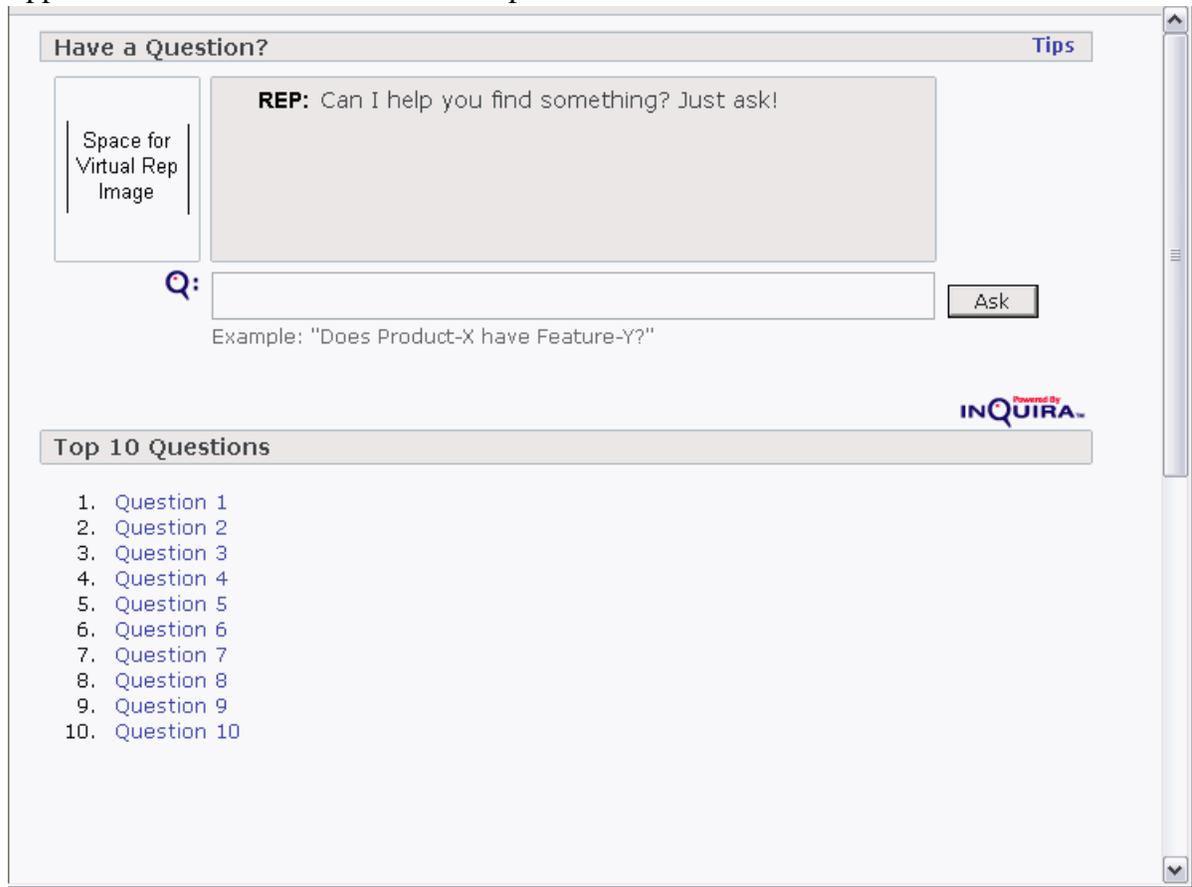
The default request area provides the functional and presentation elements required for integrating a request area into pages within your web site. The request area elements are described in more detail in [Request Elements](#) on page 23.



The screenshot displays a web interface for a request area. At the top, there is a header bar with the text "Have a Question?" on the left and a "Tips" link on the right. Below this header, the text "Type your question below to find an answer now." is displayed. A search icon (a magnifying glass) is positioned to the left of a text input field. To the right of the input field is an "Ask" button. Below the input field, an example question is provided: "Example: 'Does Product-X have Feature-Y?'". In the bottom right corner of the main content area, the INQUIRA logo is visible, with the text "Powered By" above it. Below the main content area, there is a section titled "Top 10 Questions" in a grey header bar. This section contains a numbered list of ten items, each labeled "Question 1" through "Question 10". The interface is enclosed in a window-like frame with a vertical scrollbar on the right side.

Dialog Request Area Example

The dialog-style request area provides the functional and presentation elements required for integrating a dialog-style request area into pages within your web site. The request area elements are described in more detail in [Request Elements on page 23](#). See [Implementing a Virtual Representative for more information about using the dialog style template to support user interaction with a virtual representative.](#)



Response Element Templates

The response element templates determine the basic form and content of the response elements.

Standard answer elements

results.xsl

This template contains elements for presenting standard answers, and also contains the basic building blocks for answers used by all answer purposes.

Portlet answer elements

`sidebar.xsl`

This template contains elements that generate the portlet display area of the response page. The portlet display area displays all answer purposes except standard, dialog, and direct page display.

See [Default Answer Purposes on page 27](#) for information on default answer purpose presentation.

User Interface error messages

`error.xsl`

This template specifies the format for displaying error messages. This template is required.

Global Elements and Utilities

The global element templates specify basic colors, fonts, and section headings and other variables used throughout the User Interface. The utilities files include graphics directories and basic usability functions. You can specify elements within these templates for either the two- or three-column layout style.

Common elements

`includes.xsl`

This template contains the elements that support inclusion of basic style sheets and utilities, such as CSS and JavaScript.

Global Javascript file

`qna_common.js`

This is the main JavaScript file, located in `<InQaira_home>/inquire/int/js`. It contains basic JavaScript functions used on the request and response pages.

Common element style sheet

`qna_style.css`

This is the style sheet, located in `<InQaira_home>/inquire/int/js`, that defines the basic common elements, such as fonts and colors, for the request and response page elements. See [Customizing General Style Elements on page 36](#) for more information on the style elements.

Common image directory

`images/*.gif`

This directory contains various images used throughout the User Interface. It also stores custom images, such as character images for dialog-style interaction, as described in [Implementing a Virtual Representative](#).

Chapter 3 User Interface Elements

The various templates and style sheets within the User Interface define the elements that process user requests and display application responses. Request elements and response elements include both functional elements, such as the question input box, and presentation elements, such as color schemes and heading text, that organize the application functions into a meaningful visual display.

Request Elements

The functional and presentation elements of the user request area appear on the initial request page and on the response page. Request elements include the question box for user input and other functional and graphic elements.

Element	Description
Request Area	Defines the request area elements.
Request Heading	Specifies the text that appears at the top of the request area. See Customizing the Request Heading on page 44 .
Example Question	Specifies the example question text that appears below the request heading. See Customizing the Example Question on page 44 .
Question Box	Defines the text input box. See Customizing the Question Box on page 45 .
Tips Link	Specifies the link to the User Interface help page. See Customizing the Tips Link on page 45 .
Submit Button	Specifies the request submittal mechanism. See Customizing the Submit Button on page 45 .

Response Elements

The User Interface displays answers and related information on the response page. The response page is divided into several functional areas:

- The request area, which provides the means for users to ask additional questions
- The answer area, which presents the application responses that directly the user's question
- The related information area, which presents related responses, grouped into separate portlets by answer purpose

NOTE: You can also use the direct page display feature to display the document that contains the answer to a specified request directly on the response page. Direct page displays supersede the standard answers. See [Implementing Direct Page Display](#) for more information on configuring the direct page display feature.

Response elements include answers, which are composed of various configurable sub-elements, and other functional and graphic elements.

Element	Description
Answer Area	Defines the answer display area on the response page. See Customizing Response Elements on page 46 .
Question Echo	Specifies the display of the user's question on the response page. See Customizing the Question Echo on page 46 .
Answer Introduction	Specifies text that introduces the answer. See Customizing the Answer Introduction on page 47 .
Answer Heading	Specifies the format of the document titles displayed as answer headings. See Customizing Answer Headings on page 47 .
Answer Body Text	Specifies the display font for answers on the response page. See Customizing the Answer Body Text on page 47 .
Answer Document Link	Specifies the format of the link text within answers. See Customizing the Answer Document Link on page 48 .
Related Information	Specifies the format of the elements that make up the answer portlets. See Customizing Response Elements on page 46 .

Answer Display Features

The User Interface contains features that display a variety of visual cues that accompany answers. These features include:

Answer source icons

Answer source icons indicate the type of document or information source in which the answer is located. They are passed in the XML response format in a standard attribute called `docType`. The User Interface displays icons for the following answer sources:

- Answers from structured information (database) sources: 
- Answers from HTML, newsgroups, Microsoft PowerPoint, and ASCII text documents: 
- Answers from PDF documents: 
- Answers from Microsoft Word documents: 
- Answers from Microsoft Excel documents: 
- Images: 

Answer highlighting and question-word emphasis

The stylesheet `qna_style.css` contains settings to emphasize words and phrases in the answer excerpt. Various levels of emphasis are defined in the User Interface, and these levels correspond to values defined for primary and secondary word-matching and proximity to words occurring in the user's question. Matching words are determined by the language analysis process, which takes into account *word-form va*. The default setting applies a bold style (**bold**) and a blue background to matching words.

Similar answer link

The similar answer link provides access to answers derived from other pages on the site having similar content that were found in the search. This feature enables the User Interface to consolidate duplicate pages, or pages that re-use a substantial amount of content, in the initial response. Users can click on the link to display the full answer page including the similar answers, variations, synonyms, and other semantic relationships, as described in the [Intelligent Search Language Tuning Guide](#).

Answer Purposes

Answer purposes are categories to which you assign answer actions within Dictionary rules. Answer purposes correspond to display characteristics defined in the User Interface, enabling you to establish consistent, focused, and targeted presentation for various types of application content, such as general site information, online glossaries, promotional material, and site features, such as calculators and other tools.

InQuira 8.1 is installed and configured with a standard set of answer purposes, described in [Default Answer Purposes on page 27](#), which are designed for use with the Personalized Response User Interface. The default answer purposes associate each purpose with a defined response category area, or portlet, of the answer page.

You use answer purposes by:

- Assigning answer purposes to actions within Rules, as described in [Rules](#) in the [Intelligent Search Language Tuning Guide](#).
- Configuring presentation characteristics for User Interface portlets, as described in [Configuring Answer Purposes on page 48](#).

NOTE: In contrast with answer purposes, answer methods correspond to type of data or method used to supply the answer. Examples of answer methods include querying structured data, searching the indexed unstructured content, and displaying custom content. See [Answer Action Methods for Rules](#) in the [Intelligent Search Optimization Guide](#) for more information on answer methods.

Default Answer Purposes

The standard set of answer purposes described below are designed for use with the Personalized Response User Interface.

Purpose	Description	Default Response Template	Default Presentation
Answer	Displays responses that directly address the user's question.	Answer Template	In the Answer area of the response page
Act	Displays links that provide actions that the user can take on the web site.	Act Template	In the Act Now portlet
Promote	Displays cross-sell or up-sell advertisements for products related to the intent of the question.	Promote Template	In the Promotion portlet
Related Topic	Displays links to major topic categories defined for the web site.	Link To Category Template	In the Related Topics portlet
Define	Displays links to terms used in the question as well as similar content.	n/a	In the Definition portlet
Jump to Page	Displays content configured in the Dictionary for use with the direct page display feature.	n/a	See Implementing Direct Page Display on page 61.
Converse	Displays conversational response intended for use with a virtual representative on the response page.	Converse Template	See Implementing a Virtual Representative on page 63.
Feature Content	Displays specific featured content from the web site that supplements the answers.	Feature Content Template	In the Featured Content area of the response page
Contact	For use with the Contact Deflection feature.	n/a	See the section on Implementing Contact Deflection for Web-based Email .

Answer Portlets

User Interface portlets are defined regions of the answer page. Portlets enable you to categorize responses displayed on the answer page according to purpose; some desirable responses are direct answers to user questions, while others might be information about related promotions, services, tools, and terms.

The User Interface is installed with a set of default portlets that correspond to the purposes that you can specify specific responses within the application Dictionary.

In general, the User Interface portlets are designed to accept and present information associated with any type of answer action that can be specified within a Rule; however, this section does describe limitations and suggested applications where appropriate.

See the [Intelligent Search Language Tuning Guide](#) and [Intelligent Search Optimization Guide](#) for more information on the Dictionary, Rules, actions and answer purposes and methods.

Default Answer Portlets

The User Interface is installed with several pre-defined portlets. Each portlet is designed to present answers with a specific purpose, as described in [Answer Purposes on page 26](#)

The following table describes the available default portlets. The default answer page displays the portlets in a single column to the right of the answer area. The portlets are listed here in the order in which they are displayed in the default User Interface.

Portlet	Usage
Promotions	Use this portlet to display promotional information, such as cross-sell or up-sell advertisements for products related to the intent of the question. You can configure responses to include graphics as links to pages that contain more detailed information. See Promotions Portlet Example on page 30 for more information.
Act Now	Use this portlet to display information about relevant activities that users can perform immediately on the site. This portlet favors concise, imperative messages that compel users to access beneficial features. See Act Now Portlet Example on page 31 for more information.
Learn More	Use this portlet to display brief summaries of content areas that are relevant to the user's question, such as tools and calculators. See Learn More Portlet Example on page 32 for more information.
Definition	Use this portlet to display definitions of terms related to the user's question. This portlet is ideal for displaying existing glossary information adapted from various formats. See Definition Portlet Example on page 33 for more information.
Feature Content	Use this portlet to display more detailed information about relevant content areas and site features, such as tools and calculators. The Feature Content portlet displays responses in the lower portion of the answer area and not in a segregated box, which provides space for more detailed information, such as graphical tools. See Feature Content Portlet Example on page 34 for more information.

Promotions Portlet Example

The Promotions portlet is intended to display relevant promotions and special offers. The Promotions portlet provides an opportunity to create effective context-sensitive marketing by configuring Promotional responses based on products or services mentioned the user's question.

The Promotions portlet can display responses generated by any of the available answer methods; however, it is well-suited to present custom content answers. You can configure a custom content response to include a graphic as in the following example:

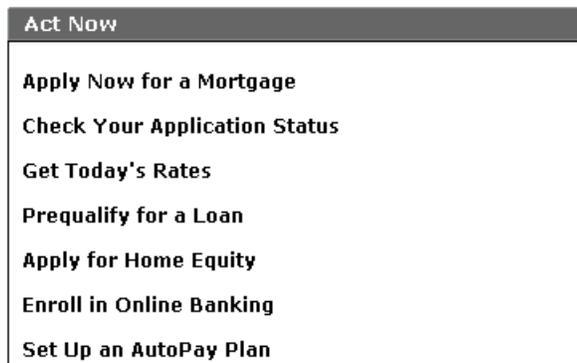


See *Intents*, *Intent Hierarchies*, *Intent Responses* in the *Intelligent Search Language Tuning Guide* for more information on configuring custom content responses.

Act Now Portlet Example

The Act Now portlet is intended to provide quick access to relevant activities that users can perform on your site. Opening an account, registering for a service, and checking the status of an order are examples of actions that you can configure as Act Now responses.

The Act Now portlet can display responses generated by any of the available answer methods; however, it is well-suited to present custom content answers that specify a title as a link to the desired location, as in the following example:



See [Intents](#), [Intent Hierarchies](#), [Intent Responses](#) in the [Intelligent Search Language Tuning Guide](#) for more information on configuring custom content responses.

Learn More Portlet Example

The Learn More portlet is intended to provide access to related topic areas and site features. You can use the Learn More portlet to direct users to FAQ pages, process overview pages, tools and calculators, and other site resources.

The Learn More portlet can display responses generated by any of the available answer methods. It is well-suited to present:

- Custom content answers that specify a title as a link to the desired location
- Custom content responses that include additional descriptive text, as in the following example:

Related Links
Mortgage Loan Process
Online Mortgage Answers
Mortgage Calculators
Home Equity Loan A Home Equity Loan provides flexible terms with a fixed monthly payment to allow for easy budgeting.
Office Locator

See *Intents*, *Intent Hierarchies*, *Intent Responses* in the *Intelligent Search Language Tuning Guide* for more information on configuring custom content responses.

Definition Portlet Example

The Definitions portlet is intended to present glossary information that may or may not be accessible on the site. InQuira 8.1 uses a special Dictionary component called an alias list to store glossary information for use by the application. The application then generates a Definitions response whenever a configured glossary term occurs in a question.

The default Definitions portlet displays the glossary term as a link that users can click to display the associated definition on a separate answer page.

The Definitions portlet is recommended for use with the Glossary answer purpose as in the following example:

Definitions

APR
The annual percentage rate is defined as the cost of money borrowed expressed as an annual rate. It's a way for consumers to compare similar mortgages from different lenders. Costs used to calculate APR are mortgage interest, discount points or origination fees, lender fees such as underwriting, processing document preparation, mortgage broker fees and private mortgage insurance fees.

Adjustable Rate Mortgages
An adjustable rate mortgage (ARM) is a mortgage for which the interest rate is not fixed but changes during the life of the loan. Lenders generally charge lower initial interest rates for ARMs than for fixed-rate mortgages. Additionally, you may be able to qualify for a greater amount under an ARM program than a fixed-rate program.

Closing Costs
Costs included in this calculation are the appraisal fees, underwriting or other lender fees, title insurance and escrow fees.

See [Glossary Answer Action Method](#) in the *Intelligent Search Optimization Guide* for more information on accessing glossary information.

Feature Content Portlet Example

The Feature Content portlet is intended to direct users to site features and resources. The Feature Content portlet is similar in intent to the Learn More portlet; however, the default User Interface displays Feature Content responses inline with the standard answers, enabling more information to be displayed for each response.

The Feature Content portlet can display responses generated by any of the available answer methods; however, it is well-suited to present custom content answers that include HTML-based functionality, as in the following example:

== FEATURE CONTENT ==

☰ Is refinancing right for me?

Am I better off refinancing?

Inputs	Results	Graphs	Tables	Help
--------	---------	--------	--------	------

Regarding your current loan

Original loan amount	\$	100,000
? Original term (years)		30
Years already paid		3
? Balloon year (zero if none)		0
? Interest rate		8.000 %

Chapter 4 Customizing the User Interface

The basic customization tasks for integrating the User Interface include:

- Specifying the layout style within the main template
- Customizing common elements, such as fonts, background colors, and graphic images, as described in [Customizing Style Elements on page 36](#)
- Customizing request and response elements, as described in [Customizing Request Elements on page 43](#) and [Customizing Response Elements on page 46](#)
- Implementing optional features, as described in [Implementing Optional Features on page 55](#).

Specifying the User Interface Layout

You specify the layout of the User Interface by specifying one of the global templates available in the extensible style language (xsl) file `main.xsl`, located in:

```
int/xsl/search/main.xsl
```

The `main.xsl` file contains include statements for the basic search layout and the additional personalized navigation and virtual representative features. Each include statement refers to one of the available global templates:

Layout Style	Description
<code>ui_search_basic.xsl</code>	Specifies the basic User Interface layout. This statement is enabled by default.
<code>ui_search_and_browse.xsl</code>	Specifies to display the Personalized Navigation user interface elements as described in Activating the Personalized Navigation User Interface Layout on page 59 .
<code>ui_search_vrep.xsl</code>	Specifies to display the virtual representative user interface elements as described in Implementing a Virtual Representative on page 63 .

IMPORTANT: You can enable only one of the include statements for your application.

The following example shows the default implementation, which enables the basic search layout, `ui_search_basic.xml`:

```
<!-- Options for Search UI Main Screens -->
<xsl:import href="ui_search_basic.xml"/>
<!--xsl:import href="ui_search_and_browse.xml"/-->
<!--xsl:import href="ui_search_vrep.xml"/-->
```

Integrating the User Interface

To integrate the InQuira 8.1 User Interface with your web site, you need to

- Integrate standard elements from your site, such as navigation and graphics, into the selected layout template
- Reference the URL of the installed and customized User Interface layout template from the appropriate locations within your site pages, such as search boxes and relevant navigation links

Customizing Style Elements

You can customize style elements of the User Interface, such as fonts, background colors, and margins, by modifying the values contained in the User Interface stylesheet, `qna_style.css`. The stylesheet defines presentation for general elements used in multiple locations, and sets of related elements, as described in:

- [Customizing General Style Elements on page 36](#)
- [Customizing Question Area Definitions on page 38](#)
- [Customizing Answer Area Definitions on page 39](#)
- [Customizing Sidebar Area Definitions on page 41](#)

Customizing General Style Elements

The general style elements determine style and formatting of various elements used throughout the user interface.

Element Name	Description
--------------	-------------

qna-page-body	This element defines the properties for the HTML <code><body></code> element of the page. It establishes general settings for relative font size, font type, page color, and page margins by default.
qna-normal-text	This element defines generic properties for a variety of text on the page. It establishes font type and relative font size.
qna-small-text	This element defines generic properties for a variety of text on the page, similar to <code><qna-normal-text></code> . It establishes font type and relative font size for text elements intended to be a little smaller than normal, such as the text just above the question box.
qna-input-textarea	This element defines properties for the HTML <code><textarea></code> object used for the question box. It establishes font type, relative font size, and scrolling properties.
qna-input-button	This element defines properties for HTML form buttons on the page. By default, it sets the font type, size and color as well as the button color.
qna-header	This element defines properties for the question box and top ten questions header bars on the main search page. It sets the font type, weight, and color, as well as the background color of the header bar.
qna-header-side	This element defines properties for headers in the left sidebar when using the <code>question_and_results_side.xsl</code> template, similar to the <code><qna-header></code> element.
qna-field-label	This element defines properties for the label next to the question box. By default, it sets the alignment to the top-right of its table cell.
qna-help-link	This element defines properties for the link to the help (Tips) page located near the question box. It sets the font type, size, color and alignment within its area.
qna-radio-link	This element defines properties for the links around the user feedback options. It sets the font type and color, alignment within its area, and underline properties to distinguish the options from hyperlinks.
qna-similar-link	This element defines properties for the link text to similar answers. It sets the font type and color by default.
qna-link	This element defines properties for general purpose link objects such as the paging links. It sets the font type, color, and alignment.
qna-area-separator	This element defines properties for any lines used to separate major sections of the User Interface. By default, it is used to draw the line between the results and the site navigation. It is primarily used in the main User Interface integration files such as <code>question_and_results.xsl</code> and only defines a background color by default.
qna-area-separator-dark	Same as <code><qna-area-separator></code> , but used for a second level of separation in some cases, such as between results and the sidebar.
qna-header-separator	This element defines properties for the lines around header bars such as the question box on the main search page. It only defines the color of those lines by default.

qna-footer-separator	This element defines properties for the separator line at the bottom of a results page. It only defines the color of that line by default.
-----------------------------	--

Customizing Question Area Definitions

The question area elements determine style and formatting of various elements used within the User Interface question area.

Element Name	Description
qna-question-header	This element defines properties for the question box header bar that is used when shown at the top of the results list. It sets the font type, relative size and color.
qna-question-sidebar-header	This element defines properties for the question box header bar that is used when shown in the left sidebar. Similar to <code><qna-question-header></code> .
qna-question-label	This element defines properties for the label next to the repeated question on the page. It specifies the font type, weight, relative size, color and alignment. The default label text is: <code>You Asked</code> .
qna-question-text	This element defines properties for the repeated question on the page. It sets the font type, weight, relative size and color.
qna-question-sidebar-label	This element defines properties for the label next to the repeated question when shown in the left sidebar (using <code>question_and_results_side.xsl</code>). Similar to <code><qna-question-label></code> .
qna-question-sidebar-text	This element defines properties for the repeated question when shown in the left sidebar area (using <code>question_and_results_side.xsl</code>). Same as <code><qna-question-text></code> .
qna-question-sidebar-block	This element defines properties for the area where the question is repeated when using the three column layout (<code>question_and_results_side.xsl</code>). By default, it sets the background color for the area.
qna-dialog-text-question-label	This element defines properties for the label identifying the user's question when using the virtual representative interaction. By default, it sets the font family, size, weight, alignment and color for the label.
qna-dialog-text-question	This element defines properties for the user's question when using the virtual representative interaction. By default, it sets the font family, size, alignment and color for the text.
qna-dialog-text-answer-label	This element defines properties for the label identifying a virtual representative's response to the user. By default, it sets the font family, size, and weight.

qna-dialog-text-answer	This element defines properties for a virtual representative's response to the user. By default, it sets the font family, size, alignment and color for the text.
qna-dialog-sidebar-answer-label	This element defines properties for the label identifying a virtual representative's response to the user for answer labels displayed in the sidebar when using the three-column layout. Similar to <code><qna-dialog-text-answer-label></code> .
qna-dialog-sidebar-answer-text	This element defines properties for a virtual representative's response to the user for answers displayed in the sidebar area when using the three-column layout. Similar to <code><qna-dialog-text-answer></code> .
qna-dialog-image-border	This element defines properties for the border around a virtual representative's image on the screen. By default, it defines the color of the border.
qna-dialog-border	This element defines properties for the border around the text dialog between the virtual representative and the user on the screen. By default, it defines the color of the border.
qna-dialog-block	This element defines properties for the area containing the text dialog between the virtual representative and the user on the screen. By default, it defines the background color and padding for the area.
qna-example-label	This element defines properties for the label next to the question example text. By default, it defines the font family, size, color and alignment.
qna-example-label-above	This element defines properties for the label next to the question example text when it appears above the example text. By default, it defines the font family, size, color and alignment.
qna-example-text	This element defines properties for the question example text. By default, it defines the font family, size, and color.

Customizing Answer Area Definitions

The answer area elements determine style and formatting of various elements used within the User Interface answer area.

Element Name	Description
qna-result-section-header	This element defines properties for the header of each section of results (best answers, possible answers, featured content). By default, it defines the font family, weight, size, color, and alignment.
qna-result-text	This element defines high level properties for "best" answers. By default, it defines the font family, size, and alignment.
qna-result-text-small	This element defines high level properties for regular answers. By default, it defines the font family, size, and alignment.

qna-result-bar	This element defines properties for the area of the results list containing general headers and other controls. By default, it defines the font family, weight, size, and alignment.
qna-result-bar-disabled	This element defines properties for the area of the results list containing disabled controls. By default, it defines the font color.
qna-result-marker	This element defines properties for the marker identifying the beginning of an answer. By default, the marker is a document icon, but the style defines the font type, weight, size and alignment in case text elements are to be used.
qna-more-result-marker	This element defines properties for the marker identifying the more results link when shown between best and possible answers. By default, it defines the font type, weight, size, color and alignment.
qna-standard-subject	This element defines properties for the answer title. By default, it defines the font type, weight and color.
qna-standard-more-link	This element defines properties for the <code>more</code> link to the answer (if used in the design). By default, it defines the font type, weight, size, and color.
qna-standard-excerpt-block	This element defines general properties for the answer excerpt. By default, it defines the font type, size, and color, as well as spacing for the block.
qna-snippet-sentence-text	This element defines properties for the sentence in the answer excerpt that matched the user's question. By default, it defines the font size, weight, color, and background color.
qna-secondary-snippet-text	This element defines properties for the secondary word matches in the answer excerpt. By default, it defines the font size, weight, color, and background color.
qna-snippet-text	This element defines properties for the primary word matches in the answer excerpt. By default, it defines the font size, weight, color, and background color.
qna-standard-table-block	This element defines properties for the structured table display area. By default, it defines the font type, size, and color, as well as margins for the area.
qna-standard-source-block	This element is intended to define properties for text displaying the source URL of the answer. By default, the source is not shown. This definition sets a font type, size, style, and color as well as margins for the display block.
qna-standard-link-block	This element is used to define properties for useful links following the answer excerpt such as "similar answers". By default, the source is not shown. This definition sets a font type, size, and color as well as margins for the block.
qna-standard-sentence-block	This element defines properties for simple sentence answers such as managed answers that display custom content. By default, it defines the font type, size, and color, as well as margins for the block.

qna-exact-excerpt-block	This element defines properties for specially identified “exact” excerpts. In a default implementation, this is only applicable to exact answer definitions. By default, it defines the font type, size, and color as well as margins, padding and borders for the block.
qna-result-table	This element defines properties for the main table definition of a structured answer. By default, it defines the border style and color.
qna-result-table-header	This element defines properties for the column headers of a structured answer. By default, it defines the font type, weight, size, and color in addition to the border style and color.
qna-result-table-text	This element defines properties for a data cell of a structured answer table. By default, it defines the font type, weight, size, and color in addition to the border style and color.
qna-result-table-more	This element defines properties for the link to the entire table of a structured answer when displaying as a summary (usually in an answer list). By default, it defines the font type, weight, size, alignment, and color in addition to the border style and color.

Customizing Sidebar Area Definitions

The answer area elements determine style and formatting of various elements used within the User Interface answer area.

Element Name	Description
qna-sidebar-block	This element defines general properties for the area of the screen where the sidebar is to be displayed. By default, it defines the background color.
qna-sidebar-section-border	This element defines properties for the border around the sidebar area and/or individual components. By default, it defines the background color.
qna-sidebar-section-title	This element defines properties for the title area of a sidebar component. By default, it defines the font type, weight, size, and color as well as the background color for the title area.
qna-sidebar-section	This element generally defines properties for the content area of a sidebar component. By default, it defines the font type, size, alignment and color as well as the background color for the area.
qna-sidebar-section-center	This element defines properties for the content area of a sidebar component, similar to <code><qna-sidebar-section></code> , except that the content area is centered. By default, this is only used by the user feedback module.
qna-sidebar-subject	This element defines properties for answer titles within a sidebar component. By default, it defines the font type, weight and color. Similar to <code><qna-standard-subject></code> .

qna-sidebar-more-link	This element defines properties for the <code>more</code> link to the answer in a regular sidebar component (if used in the design). By default, it defines the font type, weight, size, and color. Similar to <code><qna-standard-more-link></code> .
qna-sidebar-excerpt-block	This element defines general properties for answer excerpts displayed in a regular sidebar component. By default, it defines the font type, size, and color, as well as spacing for the block. Similar to <code><qna-standard-excerpt-block></code> .
qna-sidebar-table-block	This element defines properties for structured table display areas within sidebar components. By default, it defines the font type, size, and color, as well as margins for the area. Similar to <code><qna-standard-table-block></code> .
qna-sidebar-source-block	This element is defines properties for text displaying the source URL of answers within sidebar components. By default, the source is not shown. This definition sets a font type, size, style, and color, as well as margins for the display block. Similar to <code><qna-standard-source-block></code> .
qna-sidebar-sentence-block	This element defines properties for simple sentence answers, such as managed answers that display custom content, within sidebar component. By default, it defines the font type, size, and color, as well as margins for the block. Similar to <code><qna-standard-sentence-block></code> .
qna-strong-sidebar-section-border	This element defines properties for a highlighted border around the sidebar area and/or individual components. Similar to <code><qna-sidebar-section-border></code> .
qna-strong-sidebar-section-title	This element defines properties for a highlighted title area of a sidebar component. Similar to <code><qna-sidebar-section-title></code> .
qna-strong-sidebar-section	This element defines properties for a highlighted content area of a sidebar component. Similar to <code><qna-sidebar-section></code> .
qna-strong-sidebar-section-center	This element defines properties for a highlighted content area of a sidebar component, similar to <code><qna-sidebar-section></code> , except that the content area is centered.
qna-strong-sidebar-subject	This element defines properties for highlighted answer titles within a sidebar component. Similar to <code><qna-sidebar-subject></code> .
qna-strong-sidebar-more-link	This element defines properties for a highlighted <code>more</code> link within a sidebar component. Similar to <code><qna-sidebar-more-link></code> .
qna-strong-sidebar-excerpt-block	This element defines general properties for highlighted answer excerpts displayed in a sidebar component. Similar to <code><qna-sidebar-excerpt-block></code> .
qna-strong-sidebar-table-block	This element defines properties for highlighted structured table display areas within sidebar components. Similar to <code><qna-sidebar-table-block></code> .

qna-strong-sidebar-source-block	This element is defines properties for text displaying highlighted answer source URLs within sidebar components. Similar to <code><qna-sidebar-source-block></code> .
qna-strong-sidebar-sentence-block	This element defines properties for highlighted simple sentence answers, such as managed answers that display custom content, within sidebar components. Similar to <code><qna-sidebar-sentence-block></code> .

Customizing Request Elements

The User Interface request area contains the following elements, each of which has one or more configurable properties, as described in the following sections:

- [Customizing the Request Heading on page 44](#)
- [Customizing the Question Box on page 45](#)
- [Customizing the Tips Link on page 45](#)
- [Customizing the Submit Button on page 45](#)

Customizing the Request Heading

The request heading contains the following configurable properties:

Property	Template	Element Name	Default Value
question area header	config.xml	question-area-label	Ask a Question
		question-sidebar-area-header	Ask Another Question
text to display above the question box	config.xml	question-box-header	Have a question? Type it below to find an answer now.
font characteristics	qna_style.css	qna-question-*	See Customizing Question Area Definitions on page 38

Customizing the Example Question

The example question contains the following configurable properties:

Property	Template	Element Name	Default Value
text to display below the request heading	config.xml	question-example	"Does Product X have Feature Y?"
font characteristics	qna_style.css	qna-question-*	See Customizing Question Area Definitions on page 38

Customizing the Question Box

The question box has the following configurable properties:

Property	Template	Element Name
box size	question.xml	question-top
	question_side.xml	question-side
		question-sidebar
boundary characteristics	question.xml	question-top
	question_side.xml	question-side
		question-sidebar

Customizing the Tips Link

The Tips link has the following configurable properties:

Property	Template	Element Name	Default Value
text to display	question.xml	tips-link	Tips
font characteristics	qna_style.css	qna-help-link	See Customizing Question Area Definitions on page 38

Customizing the Submit Button

The Submit button has the following configurable properties:

Property	Template Location	Element Name	Default Value
text to display	question.xml	question-top	Ask
	question_side.xml	question-side	
		question-sidebar	
font characteristics	qna_style.css	qna-input-button	See Customizing Question Area Definitions on page 38

Customizing Response Elements

The User Interface response page contains the answer area and the related information (portlet) area. The answer area contains the following elements, each of which has one or more configurable properties, as described in the following sections:

- [Customizing the Question Echo on page 46](#)
- [Customizing the Answer Introduction on page 47](#)
- [Customizing Answer Headings on page 47](#)
- [Customizing the Answer Body Text on page 47](#)
- [Customizing the Answer Document Link on page 48](#)

See for information on customizing elements in the related information area.

Customizing the Question Echo

The question echo contains the following configurable properties:

Property	Template	Element Name	Default Value
echo prefix	config.xml	question-paraphrase-label	You Asked:
font characteristics	qna_style.css	qna-question-label qna-question-text	See Customizing Question Area Definitions on page 38

Customizing the Answer Introduction

The answer introduction contains the following configurable properties:

Property	Template	Element Name	Default Value
text to display as heading for highest scoring answers	config.xml	best-answers-header	Best Answers
text to display as heading for additional good answers	config.xml	good-answers-header	Answers
font characteristics	qna-style.css	qna-result-section-header	See Customizing Question Area Definitions on page 38

Customizing Answer Headings

The headings or titles for standard answer displays contain the following configurable properties:

Property	Template	Element Name	Default Value
font characteristics	qna_style.css	qna-result-section-header	See Customizing Question Area Definitions on page 38

Customizing the Answer Body Text

The text of standard answer displays contain the following configurable properties:

Property	Template	Element Name	Default Value
font characteristics	qna_style.css	qna-result-text	See Customizing Question Area Definitions on page 38

Customizing the Answer Document Link

The link to the document that contains the answer for standard answers has the following configurable properties:

Property	Template	Element Name	Default Value
document icon	results.xml	answer-marker	document type-dependent, as described in Answer Display Features on page 25
display or not	results.xml	answer-block	Display
display text	results.xml	answer-block	more
font characteristics	qna_style.css	qna-*more-link	See Customizing Question Area Definitions on page 38

Configuring Answer Purposes

The InQuira 8.1 Personalized Response User Interface is installed with a defined set of answer purposes, which are mapped to a default set of portlets, as described in [Default Answer Purposes on page 27](#). You can also add custom answer purposes to meet specific implementation requirements.

You configure answer purposes by:

- Customizing portlet presentation, as described in [Customizing Answer Portlets on page 52](#)
- (Optional) Adding answer purposes to the application, as described in [Adding Answer Purposes to the Application on page 49](#)

NOTE: Your application may include additional industry- or domain-specific answer purposes. For more information about domain-specific answer purposes, contact your InQuira account representative.

Adding Answer Purposes to the Application

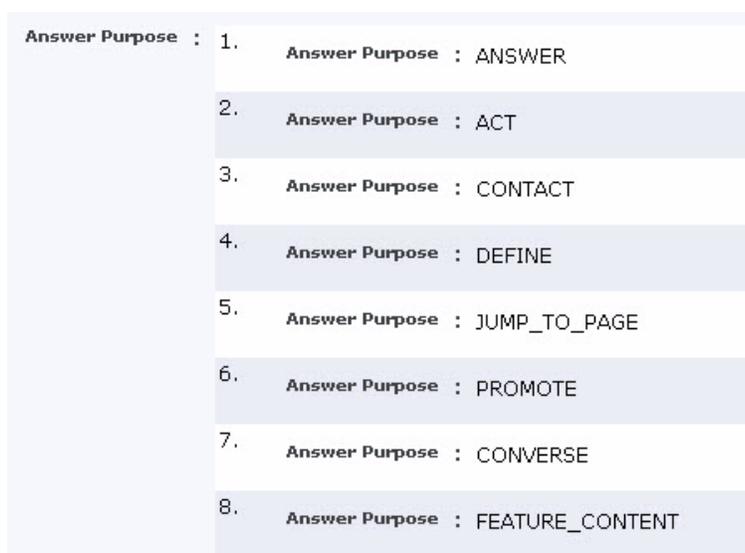
You can add and modify answer purposes in the application configuration on the Dictionary Service page of the Instances section of the Advanced Configuration Facility.

When you configure a new answer purpose, the new purpose is available to Dictionary Manager users in the Purpose drop-down menu of the Rule window.

To define or modify an answer purpose:

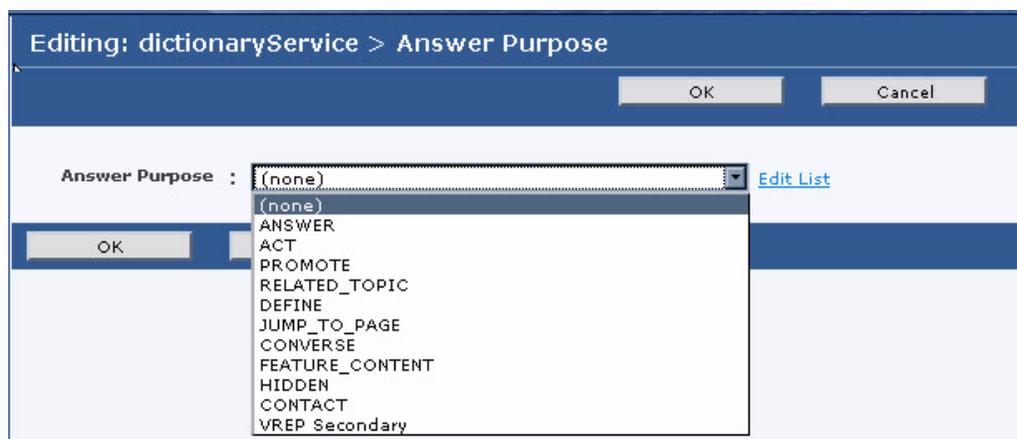
1. Select **Dictionary** from the Advanced Configuration Facility main menu.

The Answer Purpose section of the Dictionary Service page displays the currently defined answer purposes:



2. Click **Edit**.
3. Click **Add New Item** below the Answer Purpose list.

The Answer Purpose selection list displays.



4. Select **Edit List**

The Answer Purpose list displays.



5. Click an existing purpose to edit properties, or click **Add New Item** below the Answer Purpose list to create a new purpose.

The Answer Purpose page displays. The following example shows the default settings for the Answer purpose:

Description :

label :

Enabled : On Off

Max Answers ▶

Page Size ▶

Minimum Score ▶

ignore-navigation-candidates : On Off

display-area ▶

display-position ▶

Answer Template : [Edit List](#)

matched-channel :

overridenBy : (none)

[Add New Item](#)

6. Specify the following answer purpose parameters:

Parameter	Description
Description	Specify the name of the answer purpose. The name can be any alphanumeric string. Spaces and punctuation are not allowed. When the purpose is defined and enabled, this name displays in the Purpose drop-down menu of the Rule window.
Label	Specify the text to display as the portlet heading in the User Interface.
Enabled	Select On to enable this purpose. Only enabled purposes is available in the Dictionary Manager and processed by the Rules Engine and User Interface components. The default value is On. NOTE: Existing rules that specify purposes that are not enabled are processed using the Answer purpose.
Maximum Answers	Specify the maximum number of answers having this purpose to display on the response page for a given question.
Page Size	Specify the maximum number of answers having this purpose to display on the initial response page.
Minimum Score	Specify the minimum score that answers having this purpose must obtain to display on the response page for a given question. See the <i>*Intelligent Search Language Developers' Guide</i> for more information on response scoring.
Ignore Navigation Candidates	Specify whether answers having this purpose contribute to the answer totals maintained by the Personalized Navigation feature.
display-area	Specify the area of the page where the response should appear.
display-position	Specify the display position within the area. Enter a numeric value of 1-10.
Answer Template	Select an answer template from the drop-down menu to use when creating a new response.
matched-channel	Optionally, specify a channel to associate with this purpose.
Overriden By	Specify any answer purposes such that answers returned for the specified purposes are not repeated in the display for this purpose.

7. Click **OK** to save the new or modified answer purpose.

Customizing Answer Portlets

Each answer purpose that you define for your application is displayed in a separate portlet that has the following configurable elements:

- Portlet display position as described in [Specifying Portlet Display Position on page 52](#).
- Portlet headings as described in [Customizing Portlet Headings on page 52](#)
- portlet answers as described in [Customizing Portlet Answer Text on page 53](#)
- portlet document links as described in [Customizing Portlet Document Links on page 53](#)

Specifying Portlet Display Position

To specify the order in which the portlets appear on the response page, arrange the order of the portlet definition sections in the `sidebar.xml` template. Each definition section corresponds to a defined portlet. Portlets that are disabled, or for which there are no defined Rules in the Dictionary, do not display on the response page.

Sample Portlet Display Area Template provides a sample of the contents of the portlet definitions.

Customizing Portlet Headings

The answer portlet headings have the following configurable properties:

Property	Template	Element Name	Default Value
heading text	<code>config.xml</code>	*-answers-header	See Default Answer Purposes on page 27
font characteristics	<code>qna_style.css</code>	qna-(strong-)sidebar-*	See Customizing Sidebar Area Definitions on page 41
background color	<code>qna_style.css</code>	qna-(strong-)sidebar-*	See Customizing Sidebar Area Definitions on page 41

Customizing Portlet Answer Headings

The answer headings within portlets contain the following configurable properties:

Property	Template	Element Name	Default Value
font characteristics	qna_style.css	qna-(strong-)sidebar-*	See Customizing Sidebar Area Definitions on page 41

Customizing Portlet Answer Text

The answer text within portlets contain the following configurable properties:

Property	Template	Element Name	Default Value
font characteristics	qna_style.css	qna-(strong-)sidebar-excerpt-block	See Customizing Sidebar Area Definitions on page 41

Customizing Portlet Document Links

The link to the document that contains the answer for portlet answers has the following configurable properties:

Property	Template	Element Name	Default Value
display or not	results.xml	answer-block	Display
display text	results.xml	answer-block	More
font characteristics	qna_style.css	qna-*-more-link	See Customizing Sidebar Area Definitions on page 41

Chapter 5 Implementing Optional Features

The User Interface default configuration implements the standard request and response features. You can configure the User Interface to implement the following optional features:

- Process Wizards as described in *The Process Wizard User Interface* on page 56
- Personalized Navigation as described in *Activating the Personalized Navigation User Interface Layout* on page 59
- Direct page display for specified answers as described in *Implementing Direct Page Display* on page 61
- Virtual representative (VREP) dialog support as described *Implementing a Virtual Representative* on page 63
- Answer quality user feedback collection as described in *Implementing User Feedback Collection* on page 63
- Click-through logging as described in *Implementing Click-Through Logging* on page 70
- Answer highlighting within answer documents as described in *Highlighting Answers Within Documents* on page 71
- Non-English text elements as described in *Managing Multiple Languages in the User Interface* on page 75

NOTE: You can also configure the User Interface to display answers from configured Siebel 7 applications. For more information on integrating Siebel 7 applications with InQuira 8.1, see the **Intelligent Search Siebel Integration Guide*, or contact your InQuira account representative.

The Process Wizard User Interface

The Process Wizard User Interface is a set of specific pages designed for use with Process Wizards. When users select an Process Wizard answer from the standard answer page, the User Interface invokes the Process Wizard User Interface pages to display the selected Process Wizard.

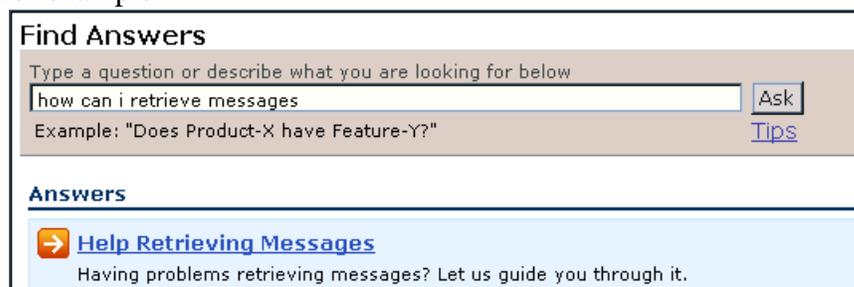
NOTE: The Process Wizard User Interface is automatically configured for use within the standard User Interface.

The Process Wizard User Interface consists of the following major elements:

- The Process Wizard answer, which displays on the answer page as described in *The Process Wizard Answer on page 56*
- The step display area, which contains the steps defined for the process, as well as the navigation buttons (Back, Next, Finish) as described in *The Step Display Area on page 57*
- The process summary column, which displays information about the previous steps that the user has taken to progress through the wizard as described in *The Step Display Area on page 57*

The Process Wizard Answer

When an end-user submits a request to the application that matches a process wizard rule, the User Interface displays a special Process Wizard answer in the standard answer area, for example:



Find Answers

Type a question or describe what you are looking for below

how can i retrieve messages

Example: "Does Product-X have Feature-Y?" [Tips](#)

Answers

[→ Help Retrieving Messages](#)
Having problems retrieving messages? Let us guide you through it.

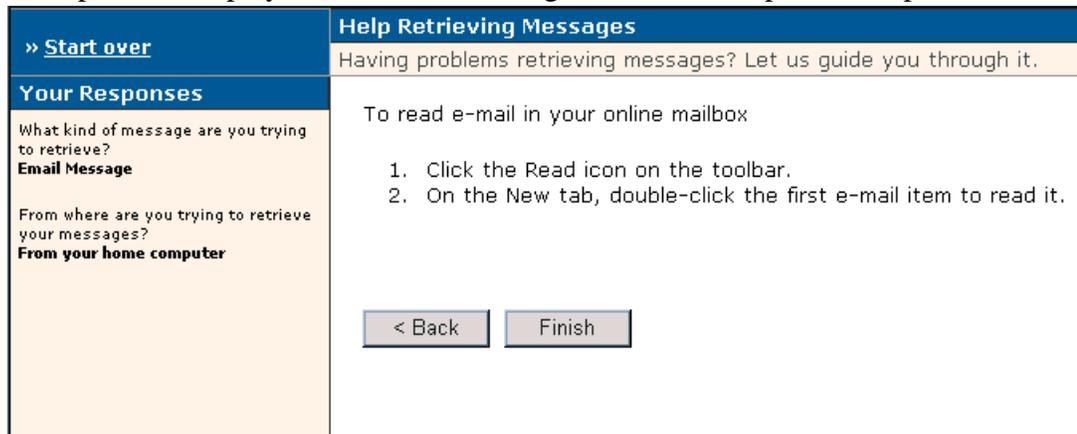
If users select the link in the process wizard answer, the User Interface displays the initial step of the process wizard.

The Step Display Area

When users select a Process Wizard answer, the application displays the initial step in the Process Wizard User Interface step display area.

The Process Wizard User Interface displays a summary of the user's previous responses to the left of the step display area. The summary displays below the heading **Your Responses**.

Each response is displayed as a link that navigates back to the process step.



Modifying the Process Wizard User Interface

You can modify Process Wizard User Interface elements to suit the needs of your application by editing the Process Wizard User Interface files, located in one of the following locations:

<InQuira_root>/inquiria/int/<subdirectory>

where:

<subdirectory> is one of the following:

- css
- js
- xsl/search

CSS Files	Description
qna_wizard_style .css	This is the style sheet that specifies the style and formatting for the elements that are specific to the Process Wizard User Interface and are not part of the standard search.

Java Script Files	Description
qna_wizard.js	This is a JavaScript library that contains Process Wizard User Interface-specific functionality.

XSL Files	Description
wizard.xsl	This is the main Process Wizard User Interface file that contains the basic page definition (similar to the ui_search* .xsl files) and utilities for the wizard pages.
wizard_fields.xsl	This file contains all of the templates used to render any defined wizard fields such as radio buttons, text boxes, select boxes, HTML areas, etc. on the UI.
wizard_history.xsl	This file contains the templates for displaying the user's choice history as well as the support templates for any actions generated by the history information such as links back to previous pages.

NOTE: The file `int/xsl/process_wizard/step.xsl` is used only for previewing steps in the Process Wizard Editor, and is not used in the Process Wizard User Interface.

Activating the Personalized Navigation User Interface Layout

To implement the Personalized Navigation User Interface elements, you activate the Personalized Navigation User Interface layout, `ui_search_and_browse.xsl`, located in:

```
int/xsl/search/main.xsl
```

The `main.xsl` file contains an include statement for the Personalized Navigation User Interface layout.

The following is an example of the include statements within the `main.xsl` file, showing the Personalized Navigation layout enabled:

```
<!-- Options for Search UI Main Screens -->
<!--xsl:import href="ui_search_basic.xsl"/-->
<xsl:import href="ui_search_and_browse.xsl"/>
<!--xsl:import href="ui_search_vrep.xsl"/-->
```

IMPORTANT: You can enable only one of the include statements for your application.

The Personalized Navigation User Interface Elements

The User Interface uses various elements to display Personalized Navigation content categories:

- Style elements, as described in [Personalized Navigation XSL Style Sheet Elements on page 60](#) and [Personalized Navigation CSS Style Sheet Elements on page 60](#)
- Resource elements as described in [Personalized Navigation-Related XML Elements on page 61](#).

Personalized Navigation XSL Style Sheet Elements

The User Interface XSL style sheets are located in:

<InQuira_home>/int/xsl/search

XSL Style Sheet	Description
ui_search_and_browse.xsl	This file is one of the main templates that determine the layout of the User Interface elements, including the question box, browse bar, answers, and sidebar. It is one of three main templates that you choose among as part of the basic User Interface implementation process as described in Specifying the User Interface Layout on page 35 .
browse_bar.xsl	This file contains the templates that render the contents of the facet navigation browse bar.
facet_table.xsl	This file contains the templates for displaying an entire table of values in response to selecting the More... link in the browse bar for categories that contain a large number of items. The More link displays a page containing all of the items.
question_browse.xsl	This file contains the definition for the question-top template used with Personalized Navigation, which differs from the standard User Interface question area.
results.xsl	This file contains updates to the standard answer block template to support facet label display within the answer section.

Personalized Navigation CSS Style Sheet Elements

The User Interface CSS style sheet is located in:

<InQuira_home>/int/css

CSS Style Sheet	Description
qna_style.css	This is the standard CSS for the User Interface. It contains new elements to support Personalized Navigation, primarily in the section labeled <code>Browse Area Definitions</code> . Additional Personalized Navigation-related definitions can be found by searching for <code>facet</code> in this file.

Personalized Navigation-Related XML Elements

The User Interface-related XML resources are located in:

<InQuira_home>/int/search

XML File	Description
resource.xml	This is the standard XML resource file, which contains new text elements and definitions. Personalized Navigation-related definitions begin with the term <code>facet-</code> .

Implementing Direct Page Display

The direct page display feature specifies direct display of the document that contains the best answer within a modified version of the response page.

The direct page display template defines an alternate response page that displays the relevant document contents in the area that the answer section would normally occupy.

The components of direct page display include:

- The Jump to Page answer purpose
- The direct page display template

You implement the direct page display feature by assigning the Jump to Page answer purpose to the appropriate Rule in the Dictionary as described in [Rules](#) in the *Intelligent Search Language Tuning Guide*.

Direct Page Display Example

The direct page display layout provides direct access to the best answer for a specified question in lieu of the standard answer display. The following example shows direct page display within a three-column layout style.

Have a Question?

Q: What is InQuira 6?

We've taken you straight to the answer we think you're looking for. If this doesn't answer your question, you can also [see more answers](#). If you're done asking questions, you can [close this frame](#).

Sign-up for up-to-date info on InQuira...

Email Address

Email this Page to:

Email Address

Company Products Markets Partners News Customers Support Careers

INQUIRA™

MARKETS & SOLUTIONS

- Financial Services
- Telecommunications
- Manufacturing
- Retail
- Interactive Marketing
- Customer Self-Service
- Employee Self-Service
- Call Center

Home : Products : InQuira 6 Overview

InQuira 6 Overview

DESCRIPTION:

InQuira 6 is a customer search and navigation application used to optimize the Web experience for prospects and customers. Utilizing dynamically assisted site navigation, InQuira 6 enables Web site question in natural language, and then interprets the real intent automatically responding with the precise answer and enriching the user to relevant content, including additional related information to buy products and services. InQuira 6 also allows users to embed rules to achieve enterprise objectives like up-sell/cross-sell and human service channels.

Ask a question

Dynamic Navigation User Interface

Managed Answers Automated Answers (Unstructured) Automated Answers (Structured)

Business Rules

NLP Search Keyword Search Question Processing Knowledge Management

Application Connectors

Data Content Packaged Apps

TOP LEVEL FEATURES & BENEFITS

Delivers best user experience:

- Managed answers
- Dynamic navigation user interface (UI)
- Natural language and keyword search
- Single point of access to structured and unstructured content

Implementing a Virtual Representative

You can configure InQuira 8.1 for virtual representative (VREP) applications. To configure an InQuira 8.1 application for use with a VREP, you need to:

- Make an image library for your VREP available to the application
- Create appropriate Dictionary rules using the Dialog answer purpose, as described in the *Rules* in the *Intelligent Search Language Tuning Guide*.
- Associate appropriate images from the library with the configured Dialog answers
- Enable the virtual representative user interface layout

The User Interface contains a dialog-style layout template, `ui_search_vrep.xml`, located in:

```
inquira/int/xsl/search/
```

The `main.xml` file contains an include statement for the virtual representative user interface layout. To enable the virtual representative user interface layout, activate the xsl import statement. The following is an example of the layout include statements showing the virtual representative layout enabled:

```
<!-- Options for Search UI Main Screens -->
<!--xsl:import href="ui_search_basic.xml"/-->
<!--xsl:import href="ui_search_and_browse.xml"/-->
<xsl:import href="ui_search_vrep.xml"/>
```

IMPORTANT: You can enable only one of the include statements for your application.

Implementing User Feedback Collection

You can collect information from customers about their satisfaction with the answers provided by the application through the user feedback feature of the response page. The user feedback mechanism consists of two components:

- The user feedback portlet as described in *The User Feedback Portlet on page 64*
- The user feedback comment page as described in *The User Feedback Comment Form on page 65*

The user feedback feature is configured by default to display in the related information area of the response page. You can disable the user feedback mechanism as described in *Disabling the User Feedback Feature on page 69*.

The User Feedback Portlet

The user feedback portlet displays by default in the related information area of the response page.



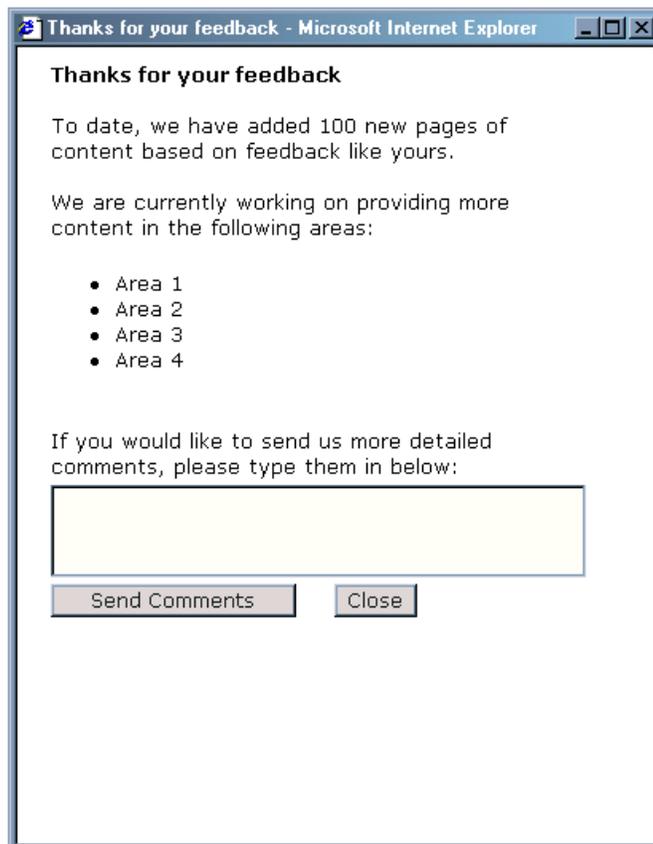
A screenshot of a user feedback portlet. It features a dark blue header with the text "Are we answering your questions?". Below the header, there are five radio button options: "Absolutely!", "Usually", "Sometimes", "Hardly", and "Not even close!". At the bottom of the portlet is a "Submit" button.

It contains the following elements that you can customize for your application:

- The user feedback heading, as described in [Customizing the User Feedback Area Heading on page 67](#)
- The rating labels, as described in [Customizing the User Feedback Rating Labels on page 68](#)

The User Feedback Comment Form

The user feedback comment form displays by default in when users submit feedback to the application.



The screenshot shows a web browser window titled "Thanks for your feedback - Microsoft Internet Explorer". The page content is as follows:

Thanks for your feedback

To date, we have added 100 new pages of content based on feedback like yours.

We are currently working on providing more content in the following areas:

- Area 1
- Area 2
- Area 3
- Area 4

If you would like to send us more detailed comments, please type them in below:

[A large, empty text input field with a light yellow background.]

[Send Comments] [Close]

The comment form provides space for users to enter additional comments. User-supplied comments are maintained in the application logs, and are available to the optional InQuira Analytics application's User Feedback report.

See [Using InQuira Analytics](#) for more information on the User Feedback report.

IMPORTANT: The default user feedback form contains sample content that is intended to be customized for your application.

The User Feedback Process

The user feedback process begins on the standard results page. The user feedback portlet solicits optional input from users. Users enter feedback by selecting from a list of radio buttons that correspond to the rating levels described in [Customizing the User Feedback Rating Labels on page 68](#).



Are we answering your questions?

Absolutely!

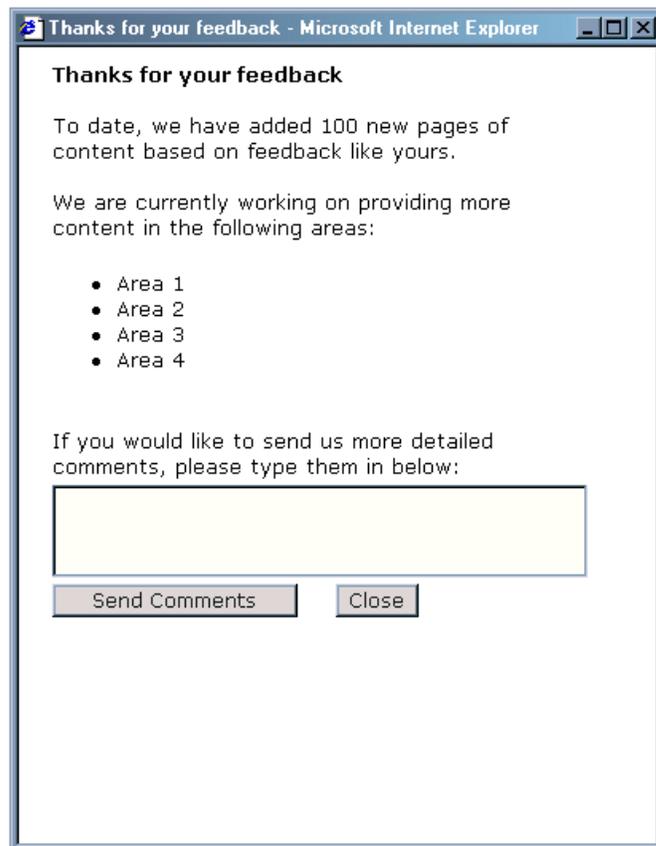
Usually

Sometimes

Hardly

Not even close!

When users submit the rating selection, the application displays the user feedback form, which must be customized for your application as described in [Customizing the User Feedback Comment Form on page 69](#). Users can enter additional feedback as text, or choose to close the feedback form.



Thanks for your feedback - Microsoft Internet Explorer

Thanks for your feedback

To date, we have added 100 new pages of content based on feedback like yours.

We are currently working on providing more content in the following areas:

- Area 1
- Area 2
- Area 3
- Area 4

If you would like to send us more detailed comments, please type them in below:

The application logs both the rating level and any optional text as a message having the identifier `ANALYTICS_USER_FEEDBACK`. The optional InQura Analytics application uses these messages to populate the User Feedback report.

Customizing the User Feedback Area Heading

You can customize the user feedback heading by editing the value specified in the User Interface resource file, `<InQura_home>/inquiria/int/xsl/search/resource.xml`.

The `resource.xml` file is divided into sections that correspond to functional areas within the User Interface.

To modify the user feedback heading:

- Locate the user feedback section, which is indicated by the label:

User feedback modules / screens

- Locate the parameter `user-feedback-header`

```
<term id="user-feedback-header">
  <entry lang="en">Are we answering your questions?</entry>
  <entry lang="de">Beantworten wir Ihre Fragen?</entry>
  <entry lang="es">?stamos contestando a sus preguntas?</entry>
  <entry lang="fr">Repondons-nous a vos questions?</entry>
  <entry lang="it">Trove le nostre risposte soddisfacenti?</entry>
  <entry lang="ja"><see original file for correct characters></entry>
</term>
```

- Edit the appropriate entry for the language of your application. For example, the default entry for English applications is:

```
<entry lang="en">Are we answering your questions?</entry>
```

Customizing the User Feedback Rating Labels

You can customize the text associated with the user feedback rating levels by editing the value specified in the User Interface resource file, `<InQuira_home>/inquiraint/xsl/search/resource.xml`.

To modify the user feedback labels:

- Locate the user feedback section, which is indicated by the label:

User feedback modules / screens

- Locate the parameter `user-feedback-rating-n`

where:

`n` is the feedback rating level. For example:

```
</term>
<term id="user-feedback-rating-5">
  <entry lang="en">Absolutely!</entry>
  <entry lang="de">Absolut!</entry>
  <entry lang="es">?bsolutamente!</entry>
  <entry lang="fr">Absolument!</entry>
  <entry lang="it">Si, assolutamente</entry>
  <entry lang="ja"><see original file for correct characters</entry>
</term>
```

- Edit the appropriate entry for the language of your application. For example, the default English rating labels are:

Rating Level	Default Value
5	Absolutely!
4	Usually
3	Sure
2	Hardly
1	Not even close!

Customizing the User Feedback Comment Form

You customize the user feedback comment form by editing the elements that control the layout and contents of the form in the user comments form style sheet,

```
<InQaira_home>/inquira/int/xsl/search/user_comments_page.xsl.
```

The user comments form style sheet is divided into sections that correspond to supported languages. For example, the section in English contains the following:

```
<b>Thanks for your feedback</b><br/>
<br/>
To date, we have added 100 new pages of content based on feedback like yours.<br/>
<br/>
We are currently working on providing more content in the following areas:<br/>
<ul>
  <li> Area 1 </li>
  <li> Area 2 </li>
  <li> Area 3 </li>
  <li> Area 4 </li>
</ul>
<br/>
If you would like to send us more detailed comments, please type them in below:<br/>
```

To modify the content and layout of the user feedback comment form:

- Locate the appropriate section for your language
- Edit the layout and content to suit the needs of your application

Disabling the User Feedback Feature

You can disable the user feedback feature by editing the User Interface configuration file,

```
<InQaira_home>/inquira/int/xsl/search/config.xsl
```

To disable the user feedback feature:

- Locate the following statement in the `config.xsl` file:

```
<xsl:variable name="get-user-feedback"...select="true()" />
```

- Change the value of the `select` parameter to `false`:

```
<xsl:variable name="get-user-feedback"...select="false()" />
```

Implementing Click-Through Logging

You can configure the User Interface to log information about the answer links selected by InQuira 8.1 users. Answer links are links to the page or document from which the application derived the answer.

When click-through logging is on, InQuira 8.1 logs a message with the identifier `ANALYTICS_CLICK_THROUGH`.

To specify click-through logging:

- Select **Click-through** from the System section of the Advanced Configuration Facility

The Click-through page displays:

Click-through

Show Advanced Options Edit

Perform HTML Highlighting : On

Perform PDF Highlighting : On

Perform Click-through Tracking : On

Perform Search Within Document : Off

Perform Default Question Highlight : Off

HTML Highlighting

Highlight Title Style : color:#000000; background:FFFF99

Highlight Sentence Style : color:#000000; background:#FFFF99

Honor Document Anchor : Off

Check For Location Replace : Off

Edit

- Select the **On** radio button in the **Perform Click-through Tracking** field
- Select **OK** to save your configuration

Highlighting Answers Within Documents

The User Interface displays links within answers that users can select to display the actual answer documents. You can configure the application to highlight the answer text within HTML and PDF documents.

You can implement document highlighting by:

- Enabling the highlighting feature
- Optionally specifying style attributes for highlighted titles and sentences within HTML documents
- Optionally specifying text string matching processes for HTML documents

IMPORTANT: The text matching algorithm and highlighting display for PDF documents is determined by the Adobe API, and is not configurable in InQuira 8.1.

Enabling Highlighting within Answer Documents

To enable highlighting for HTML and PDF documents:

- Select **Click-through** from the System section of the Advanced Configuration Facility:

The Click-through page displays:

Show Advanced Options [OK] [Cancel]

Perform HTML Highlighting : On Off

Perform PDF Highlighting : On Off

Perform Click-through Tracking : On Off

Perform Search Within Document : On Off

Perform Default Question Highlight : On Off

HTML Highlighting

Highlight Title Style :

Highlight Sentence Style :

Honor Document Anchor : On Off

Check For Location Replace : On Off

- Select the **On** radio button in the **Perform HTML Highlighting** field
- Select the **On** radio button in the **Perform PDF Highlighting** field
- Select **OK** to save your configuration

Specifying HTML Highlighting Style Attributes

You can specify HTML highlighting style attributes to apply to relevant titles and text within answer documents. You can specify any HTML statements that are valid within tags.

To specify highlighting attributes:

- Select **Click-through** from the System section of the Advanced Configuration Facility

The Click-through page displays:

Editing: Click-through

Show Advanced Options
OK
Cancel

Perform HTML Highlighting : On Off

Perform PDF Highlighting : On Off

Perform Click-through Tracking : On Off

Perform Search Within Document : On Off

Perform Default Question Highlight : On Off

HTML Highlighting

Highlight Title Style :

Highlight Sentence Style :

Honor Document Anchor : On Off

Check For Location Replace : On Off

OK
Cancel

- Enter valid style attributes in the desired fields:

Field	Description
Highlight Title Style	Specifies the style for titles within the document that match the answer text. The default style is <code>color:#000000; background:#E8F5FF</code> , which displays in standard browsers as black text on a light blue background.
Highlight Sentence Style	Specifies the style for text within the document that matches the answer text. The default style is <code>color:#000000; background:#00FF00</code> , which displays in standard browsers as black text on a bright green background.
Honor Document Anchor	Specifies that the application use existing anchors within documents to determine highlighted regions when opening the answer document in response to click-through.
Check for Location Replace	Specify this setting to check for this parameter, and display the re-directed location without performing highlighting if it is present in the answer URL. Location replace is a JavaScript mechanism used to redirect users from one page to another; however, the HTML highlighting feature cannot process the JavaScript properly.

Managing Multiple Languages in the User Interface

The User Interface is installed and configured with multi-lingual text that is stored in a resource file (`int/common/resource.xml`). The User Interface uses the language parameter to determine the appropriate text to display.

Since the default language parameter setting for the InQuira 8.1 application is English, the User Interface displays English text by default; however, setting the language parameter to another language automatically overrides the User Interface language setting.

For example, if the web server configuration or a selection mechanism on the question input page sets the language parameter to FR (French), then the User Interface displays the User Interface text element in French.

The following larger User Interface content components are also automatically translated based on the value of the language parameter

Page	Location
Tips	<code>int/xsl/tips.xml</code>
User Comments	<code>int/xsl/core/user_feedback.xml</code> (locate "template name="user-comments-page"")
Contact deflection Thank You	<code>int/xsl/contact/thank_you.xml</code>

There is no additional configuration required to implement the multi-lingual User Interface features; however, you can tailor the User Interface elements and other content to the needs of your organization by editing the referenced User Interface files.

Chapter 6 Creating a Custom Content Crawler

InQuira 8.1 includes a content acquisition framework containing base classes that support the creation of custom crawlers. The framework includes three classes: `CustomCrawlerConfig`, `CustomCrawlerConfigController`, and `CustomCrawlerState` that set up and instantiate a custom crawler.

Using the framework you can create custom content crawlers to access data from non-standard data sources and integrate it with InQuira 8.1. The example on [Example: Creating a Database Web Crawler on page 77](#), shows you how to crawl a database that tracks content on a website not otherwise crawled and consequently not available in the Content Store.

The example includes two classes: `DBWebCrawler` and `DBWebCrawlerConfig`. The `DBWebCrawler` class extends `Crawler`, the standard InQuira 8.1 class used or extended by all crawlers that do content acquisition within the content service framework. The second class, `DBWebCrawlerConfig`, shown in [Example: Configuring the Database Web Crawler on page 81](#), sets up objects used by `DBWebCrawler` and extends `CustomCrawlerConfig`.

After developing your custom crawler, continue by configuring it within the InQuira 8.1 environment as explained in the section, [Configuring a Custom Crawler on page 83](#).

Example: Creating a Database Web Crawler

The example below can be found in the file `DBWebCrawler.java`

```
package samples.content.dbwebcrawler;

import java.io.*;
import java.util.*;
import java.sql.*;
import java.net.*;

import com.inquirainfra.*;
```

```

import com.inquiria.content.*;
import com.inquiria.content.custom.*;
import com.inquiria.scheduler.job.*;
import com.inquiria.util.sql.*;

/* The DBWebCrawler class implements a custom crawler that accesses
 * a database containing URLs of documents to crawl
 */
public class DBWebCrawler
    extends Crawler
{
    protected Connection conn;
    protected Statement st;
    protected ResultSet rs;

    /* Called by the content acquisition framework prior to
     * call starting the crawl
     */
    public void connect( CrawlerConfig configuration )
        throws CrawlerException
    {
        DBWebCrawlerConfig rcc = (DBWebCrawlerConfig)configuration;

        try {
            conn = Datasource.forName( rcc.getDatasourceName( ) ).getConnection(
);CrawlerException
            st = conn.createStatement( );
            rs = st.executeQuery( rcc.getQuery( ) );

        } catch( Throwable t ) {
            throw new CrawlerException( t );
        }
    }

    /* Called by the content acquisition framework after
     * the crawl is completed
     */
    public void rundown( )
        throws CrawlerException
    {
        try {
            if( rs != null ) {
                rs.close( );
            }
            if( st != null ) {
                st.close( );
            }
            if( conn != null ) {
                conn.close( );
            }
        } catch( Throwable t ) {

```

```

        throw new CrawlerException( t );
    }
}

/* Called by the content acquisition framework prior to call starting
 * the crawl after calling connect
 */
public void start( )
{
}

/* Indicates that a single call to the findContent method discovers
 * a current document
 */
public boolean findComplete( )
{
    return true;
}

/* Indicates that this is a custom crawler */
public ContentSourceType getType( )
{
    return ContentSourceType.HTTP;
}

/* Returns all currently known document objects that are found
 * in the data source
 */
public Collection findContent( Collection priorContent,
    CrawlerConfig conf,
    CrawlerState state,
    TaskStatus status )
    throws CrawlerException
{
    Collection rc = new ArrayList( );

    try {
        String temp = null;

        while( rs.next( ) ) {
            String url = rs.getString( 1 );

            if( !rs.wasNull( ) && !url.equals( temp ) ) {
                System.out.println( "Getting URL: " + url );
                Timestamp time = rs.getTimestamp(2);

                Document d = new Document( );

                d.setCollection( conf.getCollection( ) );
                d.setFetchURL( url );
            }
        }
    }
}

```

```

        d.setDisplayURL( url );

        d.setCSType( ContentSourceType.CUSTOM );
        d.setLastModificationTime( time );
        d.setIndexingAllowed( true );
        d.setStatusCode( Document.STATUS_OK );

        rc.add( d );

        temp = url;
    } else {
        System.out.println( "NULL or Dupe!" );
    }
}
} catch( Throwable t ) {
    throw new CrawlerException( t );
}

return rc;
}

/* Returns the raw content for the given document */
public byte[] getContent( CrawlerConfig conf, Document doc )
    throws
        CrawlerException
{
    byte[] rc = null;

    URL          url    = null;
    URLConnection urlconn = null;
    InputStream  is     = null;
    ByteArrayOutputStream baos = null;

    try {
        url= new URL( doc.getFetchURL( ) );
        System.out.println( "In getContent, getting URL: " + url );
        urlconn = url.openConnection( );
        is = new BufferedInputStream( urlconn.getInputStream( ) );
        baos = new ByteArrayOutputStream( );

        byte[] buf = new byte[8192];
        int count = 0;
        while( ( count = is.read( buf, 0, buf.length ) ) > 0 ) {
            baos.write( buf, 0, count );
        }
        rc = baos.toByteArray( );

        doc.setContent( DataComponent.RAW, rc );
        doc.setDocSize( rc.length );

    } catch( ContentStoreException t ) {

```

```

        throw new CrawlerException( t );
    } catch( IOException t ) {
        throw new CrawlerException( t );
    } finally {
        if( is != null ) {
            try {
                is.close();
            } catch( IOException ex ) {
                // ignore on close
            }
        }
    }
}
return rc;
}
}

```

Example: Configuring the Database Web Crawler

This supporting class, containing configuration objects for the `DBWebCrawler` example, can be found in the file `DBWebCrawlerConfig.java`

```

package samples.content.dbwebcrawler;

import java.util.*;

import com.inquiria.content.*;
import com.inquiria.content.custom.*;

/* The CustomCrawlerConfig class implements a custom crawler configuration
 * object that knows about two non-standard configuration items:
 *
 * * datasourceName - defines the name of the data source that
 * * contains the document information
 * * query - defines the query string used to find the document information
 */
public class DBWebCrawlerConfig
    extends CustomCrawlerConfig
{
    private static final String __ident = "$Revision: 1.1.2.2 $";

    /* Compares the last modification dates of the two documents passed,
     * to determine if the document has changed
     */
    public boolean isModifiedDocument( Document currentDocument, Document newDocument )
    {
        return newDocument.getLastModificationTime().after(
            currentDocument.getLastModificationTime() );
    }
}

```

```

    }

    /* Returns the data source name */
    public String getDatasourceName( )
        throws CrawlerException
    {
        String dataSourceName = configValues.getProperty( "datasourceName" );
        if( dataSourceName == null || dataSourceName.length( ) == 0 ) {
            throw new CrawlerException( "CUSTOM_DBWEB_CRAWLER_NO_DATASOURCE",
new Object[] { getCollectionName( ) } );
        }

        return dataSourceName;
    }

    /* Returns the query string */
    public String getQuery( )
        throws CrawlerException
    {
        String query = configValues.getProperty( "query" );
        if( query == null || query.length( ) == 0 ) {
            throw new CrawlerException( "CUSTOM_DBWEB_CRAWLER_NO_QUERY", new
Object[] { getCollectionName( ) } );
        }

        return query;
    }

    /* Returns a new DBWebCrawler object */
    public Crawler getCrawler( )
        throws CrawlerException
    {
        return new DBWebCrawler( );
    }

    /* Indicates that this crawler compares existing documents in the
    * content store with documents it discovers to identify content changes
    */
    public boolean fetchExistingContent( )
    {
        return true;
    }
}

```

Configuring a Custom Crawler

The custom crawler in this example assumes that the customer has developed an in-house content publishing system that uses a database table called "content" containing a record for every document that has been published to their website. This table contains two columns:

Column Name	Description
url	This column contains the URL at which the document can be accessed on the website
modtime	This column contains the last date and time the document was published

Configuration for custom crawlers is done through the **Advanced Config** settings in the System Manager. In the example below, you can see that the `url` and `modtime` fields appear as part of the query defined in the Configuration settings in the System Manager.

To configure the custom crawler:

- Open the System Manager and choose **Advanced Config** from the Tools menu.
- Select Crawler Settings and choose **Edit**.
- Under Custom Crawlers, select **Add New Item**.
- Enter the Item Name, Class Name, and add the Configuration fields for the data source and query following the example below.

Example Crawler Settings

Item Name ▶

Custom Crawlers

Class Name ▶

Available for Unstructured Search : On Off

Document Type Detector :

Language :

Encoding :

Configuration

Item Name ▶  

Value :

Item Name ▶  

Value :

[Add New Item](#)

Chapter 7 Creating a Custom Document Preprocessor

This section shows how you can customize the way in which raw document content is processed for both text and binary files. In the example, we extend the `ProcessingFilterAdapter` class which implements the `PreprocessingFilter` interface.

The `PreprocessingFilter` interface defines the `preprocessDocument` method (text files) and `postprocessDocument` methods (text and binary files) called by `Preprocessor` when it processes content. By extending the `ProcessingFilterAdapter` class, which implements the `PreprocessingFilter` interface, we can introduce our own preprocessing and post-processing routines as part of InQuira's standard processing of text and binary files.

In the example, described in [Example: Creating a Document Preprocessor on page 85](#), we include two common preprocessing and post-processing customizations: removing footers from HTML files, and removing the table of contents from PDF files.

After developing your custom document preprocessor, continue by configuring it within the InQuira 8.1 environment as explained in the section, [Configuring a Custom Document Preprocessor on page 89](#).

Example: Creating a Document Preprocessor

The example below can be found in the file `SamplePreprocessingFilter.java`

First, we import the referenced packages.

```
package samples.prep;

import java.io.*;
import java.util.*;
import java.util.regex.*;

import com.inquiria.content.*;
import com.inquiria.prep.*;
import com.inquiria.util.xml.*;
```

Next, we set up a new custom preprocessor class by extending `PreprocessingFilterAdapter`, a class that implements the `PreprocessingFilter` interface. The `PreprocessingFilterAdapter` class adds the `getStringContent` method, which we use to get the document content for HTML files.

```

/* Implements a pre- and post-processing filter used during document conversion */
public class SamplePreprocessingFilter
    extends PreprocessingFilterAdapter
{
    /* Defines the regular expression that marks a table of contents page */
    protected Pattern tocPattern;

    /* Defines the maximum number of pages to check for table of contents */
    protected int endPage;

    /* Defines the regular expression that marks an HTML footer */
    protected Pattern footerPattern;

    /* Creates a new PreprocessingFilter instance, while configuration
    * parameters are passed in to configProperties
    */
    public SamplePreprocessingFilter( Map configProperties )
    {
        /* Assuming the configuration looks like this:
        * <preprocessingFilter name="sample">
        *   <class>samples.prep.SamplePreprocessingFilter</class>
        *   <config>
        *     <values name="hello">def</values>
        *     <values name="xyz">zyx</values>
        *   </config>
        * </preprocessingFilter>
        *
        * The Map contains entries for keys "hello" and "xyz",
        * with values "def" and "zyx" respectively.
        */
        tocPattern = Pattern.compile( "(?i)Table of Contents", Pattern.MULTILINE );
        endPage = 10;
        footerPattern = Pattern.compile( "(?i)((\u00A9|&#169;|&copy;)[ ]*)?Copyright [0-9]+ Acme,
Inc.", Pattern.MULTILINE );
    }
}

```

We first check to see if it's an HTML file, and if it is, we grab the raw file contents. We then search the contents for `footerPattern` to see if it contains footers, and if it does, we strip them out and save the contents using `setContent`.

```

/* Removes footer from HTML documents based on a regular expression */
public void preprocessDocument( Document document, CollectionConfig collection )
    throws PreprocessingException
{
    System.out.println( "preprocessDocument called for " + document.getFetchURL( ) );
}

```

```

if( document.getDocType( ).equals( DocumentType.HTML ) == true ) {
    try {
        StringBuffer rawContent = getStringContent( document );

        Matcher m = footerPattern.matcher( rawContent );
        if( m.find( ) == true ) {
            String newContent = m.replaceAll( "" );
            document.setContent( DataComponent.RAW, newContent );
        }
    } catch( ContentStoreException ex ) {
        throw new PreprocessingException( "CUSTOM_PREP_PRE_FILTER_FAILER",
            new Object[]{ document.getFetchURL( ), new Integer( document.getDocId(
                ) ) }, ex );
    }
}
}
}

```

For the TOC, we first check to see if it's a PDF file, and if it is, we grab the contents. We then search the contents for `tocPattern` to see if it contains a TOC while the page number is less than `endPage`. If we find a TOC, we strip it out and return the updated string representation of the `iqxmlNode`.

```

/* Remove Table of Contents pages from PDF documents */
public String postprocessDocument( Document document, CollectionConfig collection, Node
iqxmlNode )
    throws PreprocessingException
{
    String rc = null;

    System.out.println( "postprocessDocument called for " + document.getFetchURL( ) );

    if( document.getDocType( ).equals( DocumentType.PDF ) == true ) {
        // Since we are modifying the XML Node that represents the
        // IQXML, we need to be careful not to modify the original Node
        // unless we intentionally want to modify the XML. To signal
        // that a modification was made we need to return the string
        // representation of the new XML node that represents the IQXML
        // after post processing.
        if( removeTOC( iqxmlNode, endPage ) == true ) {
            rc = iqxmlNode.toString( );
        }
    }

    return rc;
}

protected boolean removeTOC( Node n, int lastPage )
{
    return removeTOC( n, new HashSet( ), lastPage );
}

```

```

protected boolean removeTOC( Node n, HashSet skipPages, int lastPage )
{
    boolean rc = false;
    boolean afterTOC = false;

    List children = n.getChildren( );
    if( children != null ) {
        ListIterator it = children.listIterator( );
        while( it.hasNext( ) ) {
            Object o = it.next( );
            if( o instanceof Node ) {
                Node cn = (Node)o;

                String text = cn.getText( );
                if( text != null && text.length( ) > 0 ) {
                    int pageNumber = getPageNumber( cn );
                    if( pageNumber >= 0 ) {
                        if( pageNumber >= lastPage ) {
                            afterTOC = true;
                            break;
                        }
                        Integer nPageNumber = new Integer( pageNumber );
                        if( skipPages.contains( nPageNumber ) == true ) {
                            rc = true;
                            it.remove( );
                        } else {
                            if( tocPattern.matcher( text ).find( ) == true ) {
                                rc = true;
                                it.remove( );
                                skipPages.add( nPageNumber );
                            }
                        }
                    }
                } else if( afterTOC == false ) {
                    rc |= removeTOC( cn, skipPages, lastPage );
                }
            }
        }
    }

    return rc;
}

```

We use the `getPageNumber` method in the `postprocessDocument` method to check where we are in the document.

```

protected int getPageNumber( Node n )
{
    int rc = -1;
    String auxAttr = n.getAttribute( "aux" );
    if( auxAttr != null ) {

```

```

int start = auxAttr.indexOf( " pg=" );
if( start >= 0 ) {
    start += 4;
    int end = auxAttr.indexOf( " ", start );
    if( end > 0 ) {
        rc = Integer.parseInt( auxAttr.substring( start, end ) );
    }
}
}
}
return rc;
}
}

```

Configuring a Custom Document Preprocessor

You can define configuration information for your custom document preprocessor by adding the name of the class and configuration information to the ICE `custom.xml` file as shown below. Note that you don't need to do this unless you need to pass parameters to your custom preprocessor class.

- Locate the `custom.xml` configuration file in the instance folder:
`<IS installation folder>\instances\<instance name>\custom.xml`
- Add a preprocessor node to the file as shown below substituting the class name for `samples.prep.SamplePreprocessingFilter`, and adding key value pairs as appropriate in the `<config>` section.

```

<preprocessor>
  <preprocessingFilter>
    <class>samples.prep.SamplePreprocessingFilter</class>
    <config>
      <values name="hello">def</values>
      <values name="xyz">zyx</values>
    </config>
  </preprocessingFilter>
</preprocessing>

```

Supporting Multiple Navigation Applications

To support multiple navigation applications, an entry similar to the following needs to be added to the custom.xml file:

```
<task-definition index="16">
  <name>Classification</name>
  <shortName>Navigation</shortName>
  <description>Classifies the navigation facets </description>
  <taskClass>com.inqira.navigate.ClassifyTask</taskClass>
  <parameters index="0">-p</parameters> <!-- enable progress tracking -->
  <parameters index="1">-f</parameters> <!-- applitcaion 1 name follows -->
  <parameters index="2">Default</parameters> <!-- applitcaion 1 name -->
  <parameters index="3">-f</parameters> <!-- applitcaion 2 name follows -->
  <parameters index="4">Maven</parameters> <!-- applitcaion 2 name -->
  <parameters index="5">-f</parameters> <!-- applitcaion 3 name follows -->
  <parameters index="6">Quantum</parameters> <!-- applitcaion 3 name -->
  <distributed>false</distributed>
  <needsCollection>false</needsCollection>
  <subcollection>false</subcollection>
</task-definition>
```

This overrides the default entry for the Classification task, adding the additional applications 'Maven' and 'Quantum.'

Chapter 8 Creating a Custom Task

This section shows you how to implement custom tasks to work within InQuira's system framework. The examples included here show you how to set up a simple custom task, how to handle parameters, how to display document count and progress information on the System Manager status screen, and how to set up a task so that users can interrupt it, if necessary, from the job status screen.

The following examples are provided:

- [Example: Creating a Simple Custom Task on page 92](#)
This example provides the basic template for creating a custom task.
- [Example: Handling Argument Parsing on page 94](#)
This example provides a basic template, but adds the ability to handle arguments as parameters.
- [Example: Handling Document Count and Progress Updates on page 98](#)
This example shows you how to update the document count and progress bar as documents are processed by the task.
- [Example: Handling User Task Interruptions on page 101](#)
This example shows you how to test for a request from the user to interrupt processing. Note that although we provide an example, we do not encourage you to use it unless you really need to and are able to support the consequences of interrupting a task.

After creating your custom task, continue by configuring it within the InQuira 8.1 environment as explained in the section, [Configuring a Custom Task on page 103](#).

Example: Creating a Simple Custom Task

The example below can be found in the file `CustomTaskTemplate.java`.

First, we import the referenced packages.

```
package com.inquirascheduler.job;

import org.apache.commons.cli.BasicParser;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.PatternOptionBuilder;

import org.apache.commons.cli.*;
import com.inquirascheduler.*;
import com.inquirascheduler.*;
import com.inquirainfra.*;
import com.inquiralog.*;
```

Next, we set up the new custom task class by extending `ITaskRunner` and `ILogConstants`.

```
public class CustomTaskTemplate
    implements ITaskRunner, ILogConstants {

    public void run(TaskStatus status, String[] args) throws Exception {
        boolean success = true;

        try {
```

Add the code for whatever task it is that you need to set up here

```
        /*
         * Do the actual custom task work here
         */

    }
    catch( Exception ex ) {
        //Do any appropriate logging and exception handling
        success = false;
    }
}
```

Be sure to set status here to `setStatusSuccess` if the task completes successfully, or the task defaults to `setStatusFailed`.

```
        finally {
            //Set the status at the end of the task.
            //If the status is not set, it defaults to setStatusFailed()
        }
    }
}
```

```
//causing the task and any dependent tasks to fail
if( success ) {
    status.setSuccess();
}
else {
    status.setFailed();
}
}
}

public Options getTaskOptions()
{
    Options options;

    options = new Options();

    return options;
}
}
```

Example: Handling Argument Parsing

The example below can be found in the file `CustomTaskTemplate_Args.java`.

In the first part of this custom task example, we import the referenced packages.

```

/*
 * In this custom task example, we modify it to take arguments as
 * parameters. We use the getTaskOptions() method inherited from the
 * ITaskRunner interface to handle argument parsing.
 */

package com.inquirascheduler.job;

import org.apache.commons.cli.BasicParser;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.PatternOptionBuilder;

import org.apache.commons.cli.*;
import com.inquirascheduler.*;
import com.inquirascheduler.*;
import com.inquirainfra.*;
import com.inquiralog.*;

```

Next, we set up the new custom task class by extending `ITaskRunner` and `ILogConstants`.

```

public class CustomTaskTemplate
    implements ITaskRunner, ILogConstants {

    /* Example local variables set by argument parsing */
    boolean fOption = false;
    boolean pOption = false;
    boolean rOption = false;
    String collectionName = null;

```

Here we process the `args` array and define task processing accordingly. Substitute your own switch values and parameters and processing options for ones appropriate to your task. Refer to [Configuring a Custom Task on page 103](#) for a discussion of how to handle arguments as parameters.

```

/*
 * Use a method like the one below to process the arguments
 * and set the local variables
 */
private boolean processArgs( String[] args ) {
    CommandLineParser parser;
    CommandLine commandLine;

```

```

Options optionDefinitions;
Option[] options;
boolean success = true;

parser = new BasicParser();
try {
    optionDefinitions = getTaskOptions();
    commandLine = parser.parse( optionDefinitions, args );
    options = commandLine.getOptions();

    for( int i = 0; i < options.length; i++ ) {
        List values;
        String mode;

        mode = options[i].getOpt();

        if( mode.equals( "f" ) || mode.equals( "fexample" ) ) {
            values = options[i].getValuesList();
            fOption = Boolean.valueOf( ( String )values.get( 0 ) ).booleanValue();
        }
        else if( mode.equals( "p" ) || mode.equals( "pexample" ) ) {
            values = options[i].getValuesList();
            pOption = Boolean.valueOf( ( String )values.get( 0 ) ).booleanValue();
        }
        else if( mode.equals( "r" ) || mode.equals( "rexample" ) ) {
            rOption = true;
        }
        else if( mode.equals( "c" ) || mode.equals( "collection" ) ) {
            values = options[i].getValuesList();
            collectionName = (String)values.get(0);
        }
    }
}
catch( Exception ex ) {
    //Do any appropriate logging
    Execution.context().log().event( ERROR_MSG, "CUSTOM_TASK_ERROR", ex );

    success = false;
}

return success;
}

public void run(TaskStatus status, String[] args) throws Exception {
    boolean success = true;

    try {

        if( processArguments( args ) == false ) {
            success = false;

```

```

        return;
    }

    /*
     * Do the actual custom task work here
     */

}
catch( Exception ex ) {
    //Do any appropriate logging and exception handling
    success = false;
}

```

Be sure to set status here to `setSuccess` if the task completes successfully, or the task defaults to `setFailed`.

```

    finally {
        //Set the status at the end of the task.
        //If the status is not set, it defaults to setFailed()
        //causing the task and any dependent tasks to fail
        if( success ) {
            status.setSuccess();
        }
        else {
            status.setFailed();
        }
    }
}

```

In this section we look at how to define the parameters used when the task is run.

```

/* The getTaskOptions example below shows how to define argument
 * parsing for "-p true -f false -r -c <collectionname>"
 *
 * This example uses the Apache CLI interface. Their documentation
 * can be found on their website by searching for org.apache.commons.cli
 * for their Javadoc.
 */
public Options getTaskOptions()
{
    Options options;
    Option collectionOption;
    Option pOption;
    Option fOption;
    Option rOption;

    options = new Options();

    pOption = new Option( "p", "pexample", true, "Example for an option called 'p' " );
    pOption.setArgName( "true | false " );

```

```
pOption.setOptionalArg( false );
pOption.setRequired( false );
pOption.setArgs( 1 );

fOption = new Option( "f", "fexample", true, "Example for an option called 'f' );
fOption.setArgName( "true | false" );
fOption.setOptionalArg( false );
fOption.setRequired( false );
fOption.setArgs( 1 );

rOption = new Option( "r", "rexample", true, "Example for an option called 'r'" );
rOption.setOptionalArg( true );
rOption.setRequired( false );
rOption.setArgs( 0 );

collectionOption = new Option( "c", "collection", true, "Option for the collection name." );
collectionOption.setArgName( "collection name" );
collectionOption.setOptionalArg( false );
collectionOption.setRequired( true );
collectionOption.setArgs( 1 );

options.addOption( pOption );
options.addOption( fOption );
options.addOption( rOption );
options.addOption( collectionOption );

return options;

}
}
```

Example: Handling Document Count and Progress Updates

The example below can be found in the file `CustomTaskTemplate_Prog.java`.

In the first part of this example, we import the referenced packages.

```
/*
 * This custom task template provides examples for
 * updating progress bar and document count information.
 */
```

```
package com.inquirascheduler.job;

import org.apache.commons.cli.BasicParser;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.PatternOptionBuilder;

import org.apache.commons.cli.*;
import com.inquirascheduler.*;
import com.inquirascheduler.*;
import com.inquirainfra.*;
import com.inquiralog.*;
```

Next, we set up the new custom task class by extending `ITaskRunner` and `ILogConstants`.

```
public class CustomTaskTemplate
    implements ITaskRunner, ILogConstants {

    public void run(TaskStatus status, String[] args) throws Exception {
        boolean success = true;

        try {
```

Add the code for your task here calling the appropriate method to update the document count and progress bar as indicated in the comments below. Note that the method should only be called by tasks that use a looping structure to process data so that a counter or progress indicator can be updated for each iteration of the loop.

```
/*
 * Do the actual custom task work here.
 */

/*
```

```

* The examples below show how to handle progress bar updates
* and document count updates for the task status screens in
* System Manager. They should be called from within tasks
* that use a looping structure, updating the doc count or
* progress bar as a loop iteration is completed.
*
* Option 1 for updating progress
* status.setProgress( value );
* This can be called periodically if
* a value of 1 - 100 is known and it
* makes sense to update progress with
* a specific value
*
* Option 2 for updating progress
* status.incrementProgress( incrementalvalue );
* This can be called periodically
* to increment the progress
* by some incremental value. If the
* progress was 35 and the value passed
* to this method is 4, the new progress
* will be 39.
*
* Option 3 for updating progress and doc count
* (used only if the task iterates over
* a set of documents once). This option can also only
* be used if the total number of documents to be processed
* is known at the beginning of the task.
*
* status.setTotalDocCount( total );
* This should be called at the beginning
* of the task, not inside the loop
*
* status.incrementDocProgress();
* This should be called from inside the loop, once
* for each document that was processed. Internally
* it will increment the counter for how many documents
* were processed, and also calculate the progress
* percentage based on the processed documents divided
* by the totalDocCount() value.
*
* Option 4 for updating doc count but not progress
* status.incrementDocCount();
* Increments the current doc count processed by 1,
* it starts at 0 at the beginning of every task.
*/
}
catch( Exception ex ) {
    //Do any appropriate logging and exception handling
    success = false;
}

```

Be sure to set status here to `setStatusSuccess` if the task completes successfully, or the task defaults to `setStatusFailed`.

```
        finally {
            //Set the status at the end of the task.
            //If the status is not set, it defaults to setStatusFailed()
            //causing the task and any dependent tasks to fail
            if( success ) {
                status.setStatusSuccess();
            }
            else {
                status.setStatusFailed();
            }
        }
    }

    public Options getTaskOptions()
    {
        Options options;

        options = new Options();

        return options;
    }
}
```

Example: Handling User Task Interruptions

The example below can be found in the file `CustomTaskTemplate_Prog.java`.

In the first part of this example, we import the referenced packages.

```

/*
 * In this custom task example we add a method you can call from
 * within a task loop to periodically check if a user has used the
 * job-status screen to request that the current task stop
 * processing and exit.
 *
 * It is up to the custom task code to do any necessary data
 * cleanup. If it cannot do this properly, it should not attempt to
 * support task interruption.
 */

package com.inquirascheduler.job;

import org.apache.commons.cli.BasicParser;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.PatternOptionBuilder;

import org.apache.commons.cli.*;
import com.inquirascheduler.*;
import com.inquirascheduler.*;
import com.inquirainfra.*;
import com.inquiralog.*;

```

Next, we set up the new custom task class by extending `ITaskRunner` and `ILogConstants`.

```

public class CustomTaskTemplate
    implements ITaskRunner, ILogConstants {

    public void run(TaskStatus status, String[] args) throws Exception {
        boolean success = true;

        try {

```

Add the code for your task here calling the `isInterrupted` method from within a loop to check whether the user has requested that the task be interrupted. Note that for the `isInterrupted` method to be useful it must be called from inside a loop as documents or other data are processed, so that it can poll for a change in status at each loop iteration.

```

/*
 * Do the actual custom task work here

```

```

    */
    /*
    * Handling user-interrupted task requests is only viable if
    * the task is structured in some form of loop where it can
    * periodically check if there is an outstanding request for
    * the task to interrupt itself. This should only
    * be done if if the custom task code can cleanly
    * interrupt its work without corrupting any data.
    */
    status.isInterrupted();
    //Check this method periodically in a loop. If it returns
    //true, then a user has used the job-status screen to request
    //that the current job/tasks stop processing and exit.

    //An interrupted task should be treated as a failed
    // task, so be sure to set success to false or
    // otherwise ensure that status.setFailed() is called
    }
    catch( Exception ex ) {
        //Do any appropriate logging and exception handling
        success = false;
    }
}

```

Be sure to set status here to `setSuccess` if the task completes successfully, or the task defaults to `setFailed`.

```

    finally {
        //Set the status at the end of the task.
        //If the status is not set, it defaults to setFailed()
        //causing the task and any dependent tasks to fail
        if( success ) {
            status.setSuccess();
        }
        else {
            status.setFailed();
        }
    }
}

public Options getTaskOptions()
{
    Options options;

    options = new Options();

    return options;
}
}

```

Configuring a Custom Task

To configure a custom task:

- Use one of the example templates to develop your custom task class.
- Save the file and class using the appropriate local naming conventions.
- Configure the placeholders for custom tasks in the `<number>.xml` file. You'll need to do this by hand as they cannot be configured through the System Manager. The supported placeholders can be found by searching for "PlaceholderTask" in `<number>.xml`, based on your particular task. The list of supported custom task placeholders include:
 - Pre content update
 - Pre document conversion
 - Pre indexing
 - Pre propagation
 - Pre synchronization
 - Post propagation/synchronization
 - Pre log loading
 - Post analytics processing (both Search and IM)
- Select the correct placeholder task and replace the `taskClass` configuration node (which is set by default to "com.inquirascheduler.job.PlaceholderTask") with the name of the newly defined class. An example is shown below:

```
<task-definition index="4">
<name>Pre-Document Conversion</name>
<description>Custom task to be run before document conversion.</description>
<taskClass>com.customer.services.custom.NewTask</taskClass>
<distributed>false</distributed>
<needsCollection>false</needsCollection>
<subcollection>false</subcollection>
</task-definition>
```

Other than `taskClass`, no other configuration nodes should be modified unless parameters are required.

- Compile the custom class and store it in the appropriate `services.jar` file so that ICE can add it to the `classpath`. This ensures that when the scheduler runs the task, the custom code is invoked rather than the `PlaceholderTask` class.
- To add parameters to the task definition, add parameter nodes as shown in the example below. For example, to add "-p true", "-f false", and "-r" as parameters you would add the following parameter nodes:

```

<parameters index="0">-p</parameters>
<parameters index="1">>true</parameters>
<parameters index="2">-f</parameters>
<parameters index="3">>false</parameters>
<parameters index="4">-r</parameters>

```

- If the collection name is a required parameter, set the `<needsCollection>` node to "true" and the last parameter specified to "-c". If you follow this convention the scheduler automatically adds the collection name to the arguments passed into the task. A sample result is shown below. Note that -c is the last parameter and that `needsCollection` is set to true. The `args[]` array would include the following data based on the task definition below when the task is run:

```

args[0] = "-p"
args[1] = "true"
args[2] = "-f"
args[3] = "false"
args[4] = "-r"
args[5] = "-c"
args[6] = "<collectionname>"

```

```

<task-definition index="4">
<name>Pre-Document Conversion</name>
<description>Custom task to be run before document conversion.</description>
<taskClass>com.customer.services.custom.NewTask</taskClass>
<parameters index="0">-p</parameters>
<parameters index="1">>true</parameters>
<parameters index="2">-f</parameters>
<parameters index="3">>false</parameters>
<parameters index="4">-r</parameters>
<parameters index="5">-c</parameters>
<distributed>>false</distributed>
<needsCollection>>true</needsCollection>
<subcollection>>false</subcollection>
</task-definition>

```

NOTE: When you set `<needsCollection>true</needsCollection>` thereby requiring a collection, it also dictates that the task runs once for each collection defined in the job definition. Therefore, the last parameter is a new collection name each time the task is run.

Chapter 9 Creating a Custom Authentication Interface

The default InQira 8.1 authentication interface uses Lightweight Directory Access Protocol (LDAP) to verify user access to InQira 8.1 modules. In some cases, you may want to bypass the default authentication implementation to, for example, access user information stored in a database.

The following examples provide instructions for:

- [Example: Creating a Simple Custom Authenticator on page 106](#)
Creating a basic custom authenticator built on the `IAAuthenticator` interface.
- [Example: Simple Unit Testing of a Custom Authenticator on page 108](#)
Unit-testing a custom authenticator.
- [Example: Configuration-based Test for `IAAuthenticator` Objects on page 110](#)
Testing the configured security service (IAAS).

For the code to compile, you need to download both the file for the specific authenticator and the file `TestBase.java`, which contains some of the classes called by the authenticators.

After creating your custom authenticator, continue by configuring it within the InQira 8.1 environment as explained in the section, [Configuring a Custom Authenticator on page 111](#).

Example: Creating a Simple Custom Authenticator

The example below can be found in the file `TestAuthenticator.java`, and the shared authenticator classes can be found in `TestBase.java`.

We import the referenced packages, including `TestBase.java`, which contains the shared classes referenced by the authenticator examples.

```
package samples.security.authentication;

import java.util.*;
import java.security.*;

import com.inquiria.infra.*;
import com.inquiria.infra.security.*;
import com.inquiria.infra.security.impl.*;
import com.inquiria.util.security.*;
```

We implement `TestAuthenticator` and get the user ID and password

```
/* This is a sample implementation of an authenticator */
public class TestAuthenticator
    extends TestBase
    implements IAuthenticator
{
    private static final String __ident = "$Revision: 1.1.2.1 $";

    protected String domain = "Test";

    protected Field[] authenticationFields = new Field[] {InputField
        new InputField( IFieldNames.FIELD_USER_ID ),
        new InputField( IFieldNames.FIELD_PASSWORD, true ) };

    public IUser authenticate( FieldValue[] userInfo, Map roles2PermissionsMap, long timestamp )
        throws InquiriaAuthenticationException
    {
        IUser rc = null;

        System.out.println( "TestAuthenticator.authenticate: called" );

        String userId = getFieldValue( IFieldNames.FIELD_USER_ID, userInfo );

        System.out.println( "TestAuthenticator.authenticate: userId: " + userId );

        if( userId != null ) {
            String password = getFieldValue( IFieldNames.FIELD_PASSWORD, userInfo );

            System.out.println( "TestAuthenticator.authenticate: password: " + password );
```

If the password is correct, we set up the user permissions to return using `buildUser` (defined in `TestBase.java`), and print them for test purposes. If the password is incorrect or null, we handle the exception by calling `InquiraAuthenticationException`.

```

        if( password != null && password.equals( userId ) == true ) {
            rc = buildUser( userId, domain, userInfo, roles2PermissionsMap, timestamp );
        }
    }

    if( rc == null ) {
        throw new InquiraAuthenticationException( "LOGIN_FAILED", new Object[]{ getDomain(
), userId } );
    }

    System.out.println( "TestAuthenticator.authenticate: returns: " + rc );

    return rc;
}

```

Get and return the domain (should return "Test" for the example)

```

public String getDomain( )
{
    return domain;
}

public Field[] getAuthenticationFields( )
    throws InquiraAuthenticationException
{
    return authenticationFields;
}

```

We get and print the values of authenticator, get the user's ID and password, and authenticate the user based on the ID and password. We then print out the user permissions.

```

public static void main( String[] args )
    throws Exception
{
    IAuthenticator authenticator = new TestAuthenticator( );

    System.out.println( authenticator );

    FieldValue[] userInfo = new FieldValue[]{ new FieldValue( IFieldNames.FIELD_USER_ID,
args[0] ),
                                             new FieldValue( IFieldNames.FIELD_PASSWORD, args[1] ) };

    IUser user = (IUser)authenticator.authenticate( userInfo, getRole2PermissionsMap( ),
System.currentTimeMillis( ) );
    user.dump( );
}

```

```

        System.out.println( "security keys: " + user.getSecurityKeys( ) );
        for( int i = 2; i < args.length; i++ ) {
            System.out.println( "has access to " + args[i] + ": " + user.hasAccess( new
com.inquirainfra.security.ContentPermission( args[i] ) ) );
        }
    }
}

```

Example: Simple Unit Testing of a Custom Authenticator

The example below can be found in the file `AuthenticatorTest.java`, and the shared authenticator classes can be found in `TestBase.java`

We import the referenced packages, including `TestBase.java`, which contains the shared classes referenced by the authenticator examples.

```

package samples.security.authentication;

import java.util.*;
import java.security.Permission;

import com.inquirainfra.*;
import com.inquirainfra.security.*;
import com.inquirainfra.security.impl.*;
import com.inquirainfra.config.*;

public class AuthenticatorTest
{
    private static final String __ident = "$Revision: 1.1.2.1 $";

    public static final String ROLE_LANG_DEV      = "LanguageDevelopment";
    public static final String ROLE_LANG_ADMIN   = "LanguageAdministrator";
    public static final String ROLE_ANALYTICS_ADMIN = "AnalyticsAdministrator";
    public static final String ROLE_ADMIN       = "Administrator";

    protected static final Set USABLE_PERMISSIONS;
    public static final Map DEFAULT_ROLE_PERMISSIONS;
    static {
        HashSet tmp = new HashSet( );
        String[] allPermissions = InquirainfraPermissions.PERMISSIONS;
        for( int i = 0; i < allPermissions.length; i++ ) {
            Permission p = new StandardPermission( allPermissions[i] );
            if( p.equals( new StandardPermission( InquirainfraPermissions.USERS ) ) == false ) {
                tmp.add( p );
            }
        }
    }
}

```

```

USABLE_PERMISSIONS = Collections.unmodifiableSet( tmp );

DEFAULT_ROLE_PERMISSIONS = new HashMap( );
HashSet langDevPerm = new HashSet( );
langDevPerm.add( new StandardPermission( InqiraPermissions.DICTIONARY ) );
langDevPerm.add( new StandardPermission( InqiraPermissions.TESTING ) );
langDevPerm.add( new StandardPermission( InqiraPermissions.QUALITY_MONITOR ) );
DEFAULT_ROLE_PERMISSIONS.put( ROLE_LANG_DEV, langDevPerm );

HashSet langAdminPerm = new HashSet( langDevPerm );
langAdminPerm.add( new StandardPermission( InqiraPermissions.TOP_LAYERS ) );
langAdminPerm.add( new StandardPermission( InqiraPermissions.DOMAIN_GROUPS ) );
langAdminPerm.add( new StandardPermission( InqiraPermissions.DOMAINS ) );
langAdminPerm.add( new StandardPermission( InqiraPermissions.ONT_BUILDER ) );
langAdminPerm.add( new StandardPermission( InqiraPermissions.NAVIGATION_SETUP
));
DEFAULT_ROLE_PERMISSIONS.put( ROLE_LANG_ADMIN, langAdminPerm );

HashSet analyticsAdminPerm = new HashSet( );
analyticsAdminPerm.add( new StandardPermission(
InqiraPermissions.ANALYTICS_ADMIN ) );
DEFAULT_ROLE_PERMISSIONS.put( ROLE_ANALYTICS_ADMIN, analyticsAdminPerm
);

DEFAULT_ROLE_PERMISSIONS.put( ROLE_ADMIN, USABLE_PERMISSIONS );
}

public static void main( String[] args )
    throws Exception
{
    ArrayList l = new ArrayList( );
    IAuthenticator auth = (IAuthenticator)Execution.context( ).config( ).get( new Key( args[0] ) );
    System.out.println( auth );
    l.add( auth );

    IAAS aas = new AASImpl( l, DEFAULT_ROLE_PERMISSIONS );
    System.out.println( aas );

    FieldValue[] userInfo = new FieldValue[]{ new FieldValue( IFieldNames.FIELD_USER_ID,
args[1] ),
                                           new FieldValue( IFieldNames.FIELD_PASSWORD, args[2] ),
                                           new FieldValue( IFieldNames.FIELD_DOMAIN, args[3] ) };

    RoleBasedUser user = (RoleBasedUser) aas.login( userInfo );
    user.dump( );
}
}

```

Example: Configuration-based Test for IAuthenticator Objects

The example below can be found in the file `AASTest.java`, and the shared authenticator classes can be found in `TestBase.java`.

We import the referenced packages, including `TestBase.java`, which contains the shared classes referenced by the authenticator examples.

```
package samples.security.authentication;

import java.util.*;
import java.security.Permission;

import com.inqira.infra.*;
import com.inqira.infra.security.*;
import com.inqira.infra.security.impl.*;

/* Tests the currently configured AAS */
public class AASTest
{
    private static final String __ident = "$Revision: 1.1.2.1 $";

    public static void main( String[] args )
        throws Exception
    {
        IAAS aas = (IAAS)Execution.context( ).aas( );
        System.out.println( aas );

        FieldValue[] userInfo = new FieldValue[]{ new FieldValue( IFieldNames.FIELD_USER_ID,
args[0] ),
                                                new FieldValue( IFieldNames.FIELD_PASSWORD, args[1] ),
                                                new FieldValue( IFieldNames.FIELD_DOMAIN, args[2] ) };

        IUser user = (IUser) aas.login( userInfo );
        user.dump( );

        if( args.length > 3 ) {
            System.out.println( aas.getPermission( args[3] ) );
        }
    }
}
```

Configuring a Custom Authenticator

After creating your custom authenticator, add the name of the class to the InQuira configuration file as shown below:

- Locate the latest `<number>.xml` configuration file in the configuration folder:
`%APROOT%\development\content\data\config\default\<number>.xml`
- Open the file and search for "`<choices>`".
- Add the `customAuthenticator` element under `<choices>` as shown in the example. For the `<class>` element, replace "customAuthenticator" with the name of your custom class and add an `index` element that identifies the specific version.

```
<choices>
<customAuthenticator index="0">
<class>com.inqira.infra.security.impl.TestAuthenticator</class>
</customAuthenticator>
```

- Next, define the `customAuthenticator` as the configured `securityService` by specifying the `keyref` as shown below. Replace "choices.customAuthenticator[0]" with the name of your custom class and index.

```
<securityService>
<authenticator index="1" keyref="choices.customAuthenticator[0]" />
</securityService>
```

- Your edited file should look similar to the one shown below:

```
<serviceConfiguration name="default">
<securityService>
<authenticator index="1" keyref="choices.customAuthenticator[0]" />
</securityService>
<choices>
<customAuthenticator index="0">
<class>com.inqira.infra.security.impl.TestAuthenticator</class>
</customAuthenticator>
```


Chapter 10 Integrating an External Authentication Application

If you're using a single-sign-on application you may want to bypass the default InQuira 8.1 authentication interface to intercept the qualified user data it passes, and use that to set up user access to InQuira 8.1.

Use the examples provided in:

- [Example: Integrating a Delegation Authenticator on page 114](#)

This example shows you how to integrate an external authentication application using the `IDelegationAuthenticator` interface, which extends regular authenticating modules that integrate with single-sign-on solutions.

- [Example: Integrating a Delegation Detector on page 116](#)

This example shows you how to integrate an external authentication application using the `IDelegationDetector` interface. The `IDelegationDetector` interface, in turn, is used by the request processor to extract the user information from single-sign-on solutions.

NOTE: For the code to compile, download both the file for the specific authenticator and the file `TestBase.java`, which contains some of the classes called by the examples.

After creating your custom delegation authenticator or delegation detector, continue by configuring it within the InQuira 8.1 environment as explained in the section, [Configuring a Delegation Authenticator or Detector on page 118](#).

Example: Integrating a Delegation Authenticator

The example below can be found in the file `DelegationAuthenticator.java`

```

package samples.security.delegation;

import java.util.*;
import java.security.*;

import com.inquiria.infra.*;
import com.inquiria.infra.security.*;
import com.inquiria.infra.security.impl.*;
import com.inquiria.request.*;

import samples.security.authentication.*;

/* This class supports simple delegation authorization functionality */
public class TestDelegationAuthenticator
    extends TestBase
    implements IDelegationAuthenticator
{
    private static final String __ident = "$Revision: 1.1.2.2 $";

    protected String domain = "Delegation";
    protected Field[] authenticationFields = new Field[0];
    //Indicates that it cannot be used to display a login screen

    public IUser authenticate( FieldValue[] userInfo, Map roles2PermissionsMap, long timestamp )
        throws InquiriaAuthenticationException
    {
        // Since we only want to test delegation, we provide
        // no mechanism to authenticate a user
        // through a login screen.
        return null;
    }

    public IUser delegate( FieldValue[] userInfo, Principal principal, Map roles2PermissionsMap,
long timestamp )
        throws InquiriaAuthenticationException
    {
        IUser rc = null;

        System.out.println( "TestDelegationAuthenticator.delegate: called" );

        String userId = getFieldValue( IFieldNames.FIELD_USER_ID, userInfo );
        System.out.println( "TestDelegationAuthenticator.delegate: userId: " + userId );

        if( userId != null ) {
            rc = buildUser( userId, domain, userInfo, roles2PermissionsMap, timestamp );
        }
    }
}

```

```

    }

    System.out.println( "TestDelegationAuthenticator.delegate: returns: " + rc );

    return rc;
}

public String getDomain( )
{
    return domain;
}

public Field[] getAuthenticationFields( )
    throws InquireAuthenticationException
{
    return authenticationFields;
}

public static void main( String[] args )
    throws Exception
{
    IDelegationAuthenticator authenticator = new TestDelegationAuthenticator( );

    System.out.println( authenticator );

    FieldValue[] userInfo = new FieldValue[]{ new FieldValue( IFieldNames.FIELD_USER_ID,
args[0] ),
                                             new FieldValue( IFieldNames.FIELD_DOMAIN,
authenticator.getDomain( ) ) };

    IUser user = (IUser)authenticator.delegate( userInfo, null, getRole2PermissionsMap( ),
System.currentTimeMillis( ) );
    if( user != null ) {
        user.dump( );
    } else {
        System.out.println( "Delegation for " + args[0] + " failed" );
    }
}
}
}

```

Example: Integrating a Delegation Detector

The example below can be found in the file `DelegationDetector.java`

```
package com.inquiria.infra.security.impl;

import com.inquiria.infra.*;
import com.inquiria.infra.security.*;
import com.inquiria.request.*;

/* This class implements a simple delegation detector */
public class TestDelegationDetector
    implements IDelegationDetector
{
    private static final String __ident = "$Revision: 1.4.4.1 $";

    protected String domain;

    public TestDelegationDetector( )
    {
    }
    public FieldValue[] detectDelegation( Request request )
    {
        return new FieldValue[]{
            new FieldValue( IFieldNames.FIELD_USER_ID,
                System.getProperty( "user.name" ) ),
            new FieldValue( IFieldNames.FIELD_DOMAIN, "INQUIRA" )
        };
    }

    public FieldValue[] detectDelegation( Request request )
    {
        FieldValue[] rc = null;
        String userId = null;

        System.out.println(
            "TestDelegationAuthenticator.detectDelegation: called" );

        try {
            userId = request.getUserName( );
        } catch( Exception ex ) {
            //ignore since we don't have a valid user then
        }

        System.out.println(
            "TestDelegationAuthenticator.detectDelegation: userId: " +
            userId );
    }
}
```

```

        if( userId != null && ( userId = userId.trim( ) ).length(
) > 0 ) {
            rc = new FieldValue[] {
                new FieldValue( IFieldNames.FIELD_USER_ID,
userId ),
                new FieldValue( IFieldNames.FIELD_DOMAIN,
domain ) };
        }

        System.out.println(
"TestDelegationAuthenticator.detectDelegation: returns: " + rc
);

        return rc;
    }

    public static void main( String[] args )
        throws Exception
    {
        IDelegationAuthenticator authenticator = new
TestDelegationAuthenticator( );
        TestDelegationDetector detector = new
TestDelegationDetector( );
        detector.domain = authenticator.getDomain( );

        System.out.println( detector );
        System.out.println( authenticator );

        Request request = new Request( );
        request.setUserName( args[0] );

        FieldValue[] userInfo = detector.detectDelegation(
request );

        IUser user = (IUser)authenticator.delegate( userInfo,
null, TestDelegationAuthenticator.getRole2PermissionsMap( ),
System.currentTimeMillis( ) );
        if( user != null ) {
            user.dump( );
        } else {
            System.out.println( "delegation for " + args[0] + "
failed" );
        }
    }
}

```

Configuring a Delegation Authenticator or Detector

After creating your new delegation detector, add the name of the class to the InQuira configuration file as shown below:

- Locate the latest `<number>.xml` configuration file in the configuration folder:
`%APROOT%\development\content\data\config\default\<number>.xml`
- Open the file and search for "`<choices>`".
- Add the `<delegationDetector>` element under `<choices>` as shown below. Replace "TestDelegationDetector" with the name of your custom class and add a name element that identifies the specific version.

```
<choices>
<delegationDetector name="test">
<class>com.inquiria.infra.security.impl.TestDelegationDetector</class>
</delegationDetector>
```

- Define the `delegationDetector` in as the configured `securityService` by specifying the `keyref` as shown below. Replace "`delegationDetector[test]`" with the name of your custom class and version name.

```
<securityService>
<delegationDetector keyref="choices.delegationDetector[test]" />
</securityService>
```

- Your edited file should look similar to the one below:

```
<serviceConfiguration name="default">
<securityService>
<delegationDetector keyref="choices.delegationDetector[test]" />
</securityService>
<choices>
<delegationDetector name="test">
<class>com.inquiria.infra.security.impl.TestDelegationDetector</class>
</delegationDetector>
```

Chapter 11 Creating an Action Plugin

This section deals with how to create a plugin for use within dictionary rules. The example, [Example: Creating an Action Plugin on page 119](#), implements a class that can be used to trigger an action in a rule that calls the plugin. You can have the rule be called for every question, and then implement your own custom condition.

After creating your custom plugin, continue by configuring it within the InQira 8.1 environment as explained in the section, [Configuring an Action Plugin on page 121](#).

Example: Creating an Action Plugin

The example below can be found in the file `ActionGeneratorPlugin.java`.

In the first part of this example, we import the referenced packages and display the copyright notices.

```
package com.CLIENT_NAME.inqira.action;

import com.inqira.dictionary.rules.userdata.AnswerPart
import com.inqira.dictionary.DictionaryObjectTypes;
import com.inqira.dictionary.answerlayout.AnswerPart;
import com.inqira.dictionary.answerlayout.FacetRestriction;
import com.inqira.dictionary.dictobjs.ActionRule;
import com.inqira.evaluator.Action;
import com.inqira.evaluator.ActionGenerator;
import com.inqira.evaluator.SetFacetRestrictionAction;
import com.inqira.infra.Execution;
import com.inqira.infra.InqiraException;
import com.inqira.intents.*;
import com.inqira.match.Matcher;
import com.inqira.match.SentenceMatcher;
import com.inqira.match.VariableInstantiation;
import com.inqira.match.expression.IMLExpression;
import com.inqira.nlp.Sentence;
import com.inqira.request.RequestContext;
```

```

/*
 * This class can be used to trigger an action in a rule that calls
 * this plugin. You can have the rule be called for every question, and
 * then implement your own custom condition below.
 */

```

Next, we implement the action plugin based on the ActionGenerator interface.

```

public class MyPluginActionGenerator
    implements ActionGenerator {

    /* This method is triggered when the plugin fires based on rules */
    public Action[] generate(RequestContext requestContext, AnswerPart answerPart,
        Sentence sentence,
        VariableInstantiation variableInstantiation, Map map) throws
        InquireException {

        IntentService is = Execution.context().intents();
        Action[] actions = new Action[0];

        //Test for condition to trigger the action you want

        actions = new Action[1];

```

In this example, we set up a facet restriction as our rule-based action. By setting up and defining your own action as `actions[0]` below, you can use this example to trigger other actions.

```

    /* Example
     * FacetRestriction fr = new FacetRestriction("\"CRID.\" +
     * contentRecordID.toUpperCase() + "\", true);
     * actions[0] = new PluginExactSearchAction(answerPart, fr, "FACET" +
     * facetIML);
     */

    return actions;
}
}

```

Configuring an Action Plugin

After creating your action plugin, add the name of the class to the InQuira configuration file as shown below:

- Locate the latest `<number>.xml` configuration file in the configuration folder:
`%APROOT%\development\content\data\config\default\<number>.xml`
- Open the file and search for "`<PluggableConsequences>`".
- Under `<PluggableConsequences>` add a new section like the one shown below:

```
<Consequence name="Followup">  
  <description>Module for recreating followup questions</description>  
  <class> com.inquira.analysis.followup.FollowupActionGenerator </class>  
  <parameter index="0"> type </parameter>  
</Consequence>
```
- Enter the plugin name, description, class name, and parameter for the new plugin. The parameter ("type" in this case), appears as text in a text box in the Workbench when you choose the plugin.
- Once the plugin has been added to the configuration file, it appears as a selection in the Plugin drop-down list when you set up a rule in the Dictionary Manager. For information on how to set up plugins as answer actions for rules, refer to [Advanced Features of Rules](#) in the *Intelligent Search Optimization Guide*.

Chapter 12 Creating a Custom Preference Handler

This section describes how to set up a custom preference handler by creating a new Java class that extends `NamedHandler` (see the example in [Example: Creating a Preference Handler on page 123](#)).

After creating your custom preference handler, continue by configuring it within the InQira 8.1 environment as explained in the section, [Configuring a Preference Handler on page 124](#).

Example: Creating a Preference Handler

The example below can be found in the file `PreferenceHandler.java`

In the first part of this example, we import the referenced packages.

```
package samples.preferencehandler;

import com.inqira.request.*;
import com.inqira.infra.Execution;
import com.inqira.preference.*;

import java.util.regex.*;
import java.util.*;
```

Next, we set up the custom preference handler class by implementing the `NamedHandler` interface.

```
public class SamplePreferenceHandler implements NamedHandler
{

    public RequestContext handle(RequestContext rc) throws HandlerException
    {
        // Get parameters
        Properties prop = rc.getUserAgentRequestParameters();

        try {
```

```

PreferenceService prefs = Execution.context().preferences();
System.out.println("*** Got Preference ***");

//Loop through the property names
Enumeration e = prop.propertyNames();
while (e.hasMoreElements()) {
    String propName = (String)e.nextElement();
    System.out.println("*** Got prop: " + propName);
    System.out.println("*** " + propName + " has a value of: " +
prop.getProperty(propName));

        //Assign a property value referenced by a context variable of the property name
        PreferenceValue pv = prefs.setPreferenceValue(propName,
prop.getProperty(propName));
    }
}
catch(Exception ex) {
    System.err.println("!!! Error getting preferences! " + ex);
}

return rc;
}

public String getHandlerName( )
{
    return "Sample Preference";
}
}

```

Configuring a Preference Handler

After creating the custom preference handler, add the name of the class to the InQuira configuration file as shown below:

- Locate the latest `<number.xml>` configuration file in the configuration folder:
`%APROOT%\development\content\data\config\default\`
- Open the file and search for `<requests name="AnswerQuestion">`.
- In the list of classes named `<handlers>`, add your preference handler class as `index=1` renaming all the subsequent ones.

Chapter 13 Rendering Web Pages Using a Custom Agent

This section presents an example of how to render web pages using a custom transformation tool. The example is generic, in that it does not assume anything about the type of output you may want to produce. It simply sets up the gateway, retrieves the data, and does a standard XSL transformation.

In the example (*Example: Rendering a Web Page Using a Custom Agent on page 125*), we set up a client (IClient) and initialize a connection through a SOAP gateway with the InQuira 8.1 backend. Using a subclass (XMLAgent) of the class (Agent) used by InQuira 8.1, the example gets the request parameter that defines how the retrieved data is presented in the InQuira 8.1 user interface, and continues by retrieving the data. Since XMLAgent does not carry out the transformation included in Agent, the example continues by transforming the returned XML (GIML) using the standard XSL transformation.

Prior to doing the transformation, the example sets up access to a DOM node. Using the DOM node to access the returned XML data, you can substitute your own rendering algorithms to produce output other than the standard HTML produced by InQuira 8.1.

Example: Rendering a Web Page Using a Custom Agent

The source for the example below can be found in the file `xmlui.jsp`

In the first part of the example server page we set up error handling, display the copyright notice, and import the referenced packages.

```
<%@ page errorPage="error.jsp" %>
<%--
/*
 * I n Q u i r a   Copyright (c) 2002 - 2006 InQuira, Inc. All rights
 * reserved. Use or distribution without the express written consent of
 * InQuira, Inc. is not permitted and is prohibited by law.
 */
```

```
--%>
<%@ page import="java.io.*,java.util.*" %>
<%@ page
import="javax.xml.transform.*,javax.xml.transform.stream.*,javax.xml.transform.dom.*,org.w3c.d
om.*" %>
<%@ page import="com.inquiria.infra.gateway.html13.*,com.inquiria.infra.client.*" %>
```

Next, we set up a client object using the IClient interface and initialize a connection through a SOAP gateway.

```
<%!

    private static IClient client;
    private static Object lockObject = new Object();

    static {
        client = null;
    }

%>
<% //Create an IClient object to communicate with the search back end
synchronized (lockObject) {
    if (client == null) {
        System.out.println("Initializing Connection with InQuira Gateway");
        IClient configuredClient = null;
        Properties props = new Properties( );

        // Modify the values below to adjust for your environment
        String soapurl = "http://hostname:port/inquiragw/servlet/rpcrouter;
        String soapurn = "urn:inquiria";
        String timeout = null;

        // Create, configure, and connect the SOAP client
        configuredClient = new Client( );
        props.setProperty( Client.URN, soapurn );
        props.setProperty( Client.URL, soapurl );
        if( timeout != null ) {
            props.setProperty( Client.TIMEOUT, timeout );
        }

        try {
            configuredClient.setConnectionProperties( props );
            configuredClient.connect( );
        }
        catch( ClientException ex ) {
            ex.printStackTrace( );
            RuntimeException rex = new RuntimeException( "Unable to connect to client.\nReason:
" + ex.toString( ) );
            throw rex;
        }
    }
}
```

```

        client = configuredClient;
    }
}
%>

```

Using the XMLAgent subclass of Agent, we get the request parameter that defines how the retrieved data is presented in the user interface, and retrieve the data. Refer to the comments for the switches below to find out what each parameter does.

```

<%
    // Create the XMLAgent that takes the HTTP request parameters
    // and headers and create an Inquire request, call the
    // IClient.process method, and return the Inquire
    // response in a DOM node.
    Agent agent = new XMLAgent( client, request, response, config, request.getSession(true) );

    // Get the mode we are in from the HTTP request parameter
    // called "ui_mode"
    String mode = request.getParameter( Agent.HTTP_PARAM_MODE );
    Object node = null;
    if( mode == null || ( mode = mode.trim( ) ).length( ) == 0 || mode.equals(
Agent.HTTP_PARAM_MODE_INITIAL_SCREEN ) ) {
        // If there was no mode set or it was set to "initial_screen",
        // then we want to display the entry point
        // to the search application
        node = agent.processInitialScreen( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_QUESTION ) ) {
        // If mode is set to "question", then we are answering a
        // user's question
        node = agent.processQuestionMode( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_NAVIGATE ) ) {
        // If mode is set to "navigate", then we are processing a
        // user changing navigation parameters -
        // by clicking on the facet links
        node = agent.processNavigateMode( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_ANSWER ) ) {
        // If mode is set to "answer", then we are processing an
        // answer-based request, such as highlighting
        // or click-through tracking
        node = agent.processAnswerMode( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_FEEDBACK ) ) {
        // If mode is set to "feedback", then we are handling the
        // user rating the answers
        node = agent.processFeedbackMode( );
        response.setStatus(204); // No Content response
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_PAGING ) ) {
        // If mode is set to "paging", then we are displaying the
        // prior, current, or next page of answers
        // depending on the direction

```

```

        node = agent.processPagingMode( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_GETPAGE ) ) {
        // If mode is set to "get_page", then we get a static
        // page such as the search tips
        node = agent.processGetPage( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_LOGIN ) ) {
        // If mode is set to "login", then we process a login request
        node = agent.processLoginMode( );
    } else if( mode.equals( Agent.HTTP_PARAM_MODE_SEARCH_WITHIN ) ) {
        // If mode is set to "search_within", then we process a
        // search within a given document
        node = agent.processSearchWithin( );
    } else {
        // We encountered an unsupported mode
        node = agent.processInvalidMode( mode );
    }
}

```

If we managed to retrieve some data, we continue by setting up a DOM node and doing the standard transformation normally done in InQuira. Use the DOM node to access the returned XML and substitute your own transformation algorithms to generate output other than HTML.

```

    if( node != null ) {
        // If we got a response, we try to apply the standard XSL
        // transformation to generate HTML
        DOMSource xslIn = new DOMSource( (Node)node );
        StreamResult xslOut = new StreamResult( out );
        agent.assureTemplates( );
        Transformer transformer = agent.getTemplate( "QUESTION_ANSWER"
    ).newTransformer( );
        transformer.setOutputProperty( OutputKeys.ENCODING, "UTF-8" );
        transformer.transform( xslIn, xslOut );
    }
%>

```