ConcourseSuite CRM

Writing Reports How-To

June 27, 2014

Introduction

Writing reports for ConcourseSuite CRM

The goal of the reports module is to provide a robust report engine with the simplicity of a web interface. The web interface allows users to choose a pre-defined report, customize the parameters of the report, then generate a PDF online and optionally by email.

Report writing is both a creative and a technical process. This document assumes that you have a working knowledge of databases, database queries (SQL), and software applications.

The report module incorporates the JasperReports report engine. JasperReports is a powerful open source Java reporting tool that has the ability to deliver rich content onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files.

Requirements

Report writing environment

Writing reports requires the following:

1. Access to a CRM database for reference and testing; read-only access to the production database can be used, but be aware that inefficient reports can put a strain on the database server – therefore testing with a copy of the database is recommended
2. ConcourseSuite CRM sample reports (*.jrxml and *.jasper files)
3. ConcourseSuite CRM Java libraries (concursive-commons-20120530.jar and concoursesuite.jar)
4. Either Jaspersoft's iReport or Jasper Studio for Eclipse. These are desktop applications which can design and export report XML files; we recommend downloading iReport from http://community.jaspersoft.com/project/ireport-designer – this application can be used on Windows, Linux, and Mac OSX; requires that Java is installed
5. The ConcourseSuite CRM Database Schema is useful from https://www.concursive.com/show/concoursesuite/file/5253

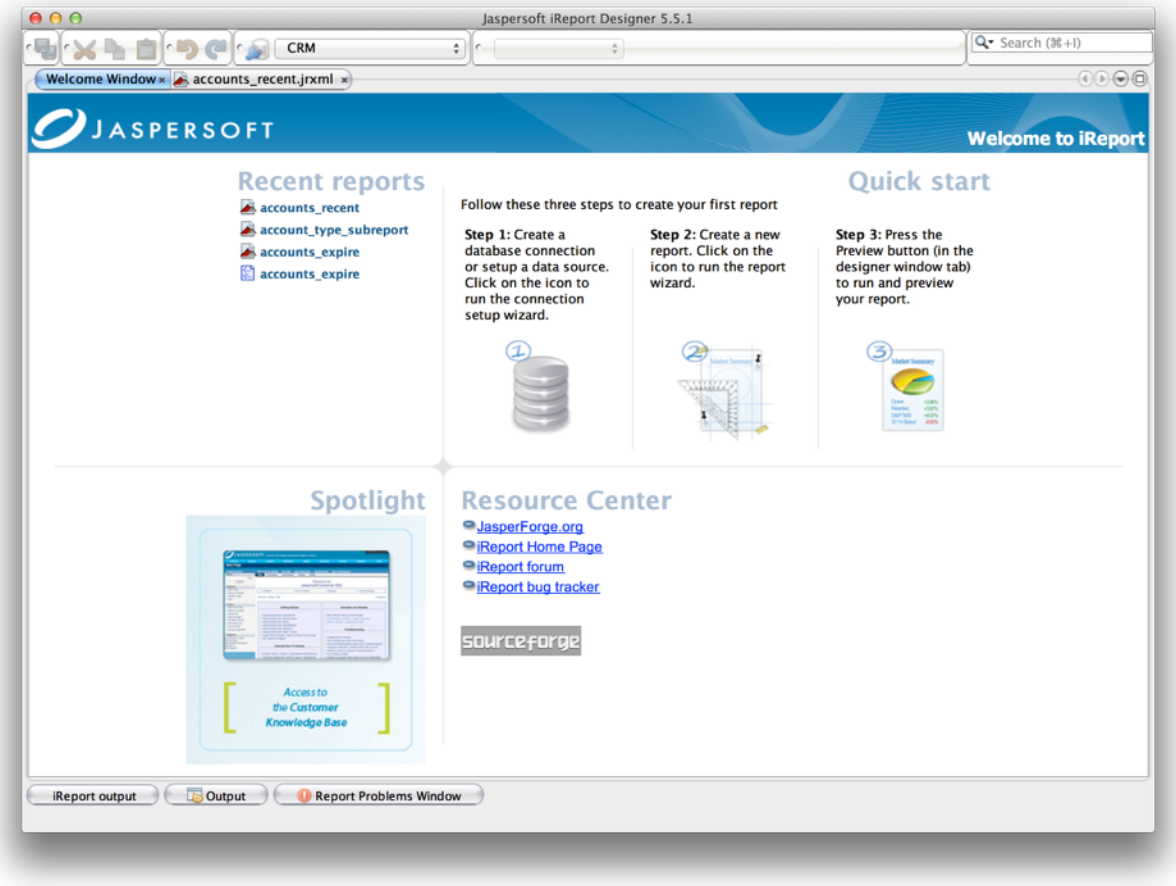
Installing iReport on your own

1. Download iReport from http://community.jaspersoft.com/project/ireport-designer by choosing the "Download" link, then choose the latest stable version, then choose to download the iReport for your operating system.
2. Unpack the downloaded iReport .zip file; you should now have an iReport directory with the iReport application
3. iReport includes the PostgreSQL database driver; other drivers may have to be downloaded separately
4. Depending on your platform, start iReport, usually by double-clicking the iReport application

iReport Configuration

Start iReport

When the application is finished loading, the main report writing window will be displayed as follows:

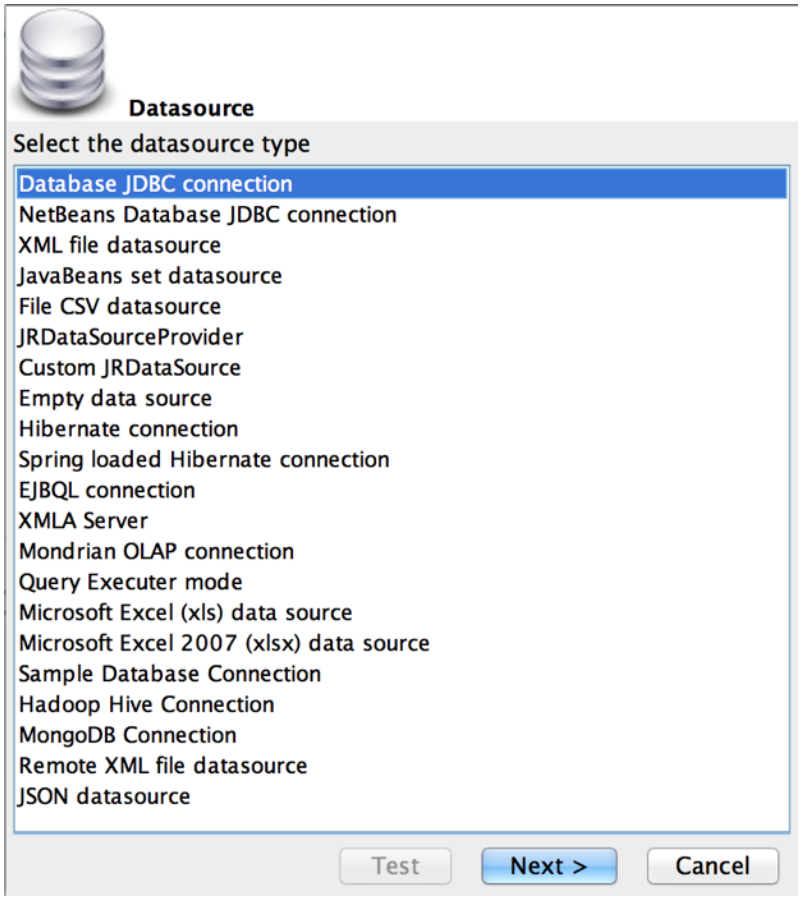


Configure Options

To configure the default settings for the application's behavior, select the "Preferences" menu item. A window will appear, and in the "General" tab, select and confirm your default units and language (this must be set to Java). In the "Compiler" tab, choose a directory for iReport to use when compiling reports and also add concoursesuite.jar and concursive-commons.jar to the iReport Classpath.

Define a Datasource

iReport will need access to either a CRM production database, or a copy of the database, in order to execute and review reports. This is done by defining a database connection to your database server. Choose “Create a database connection”.

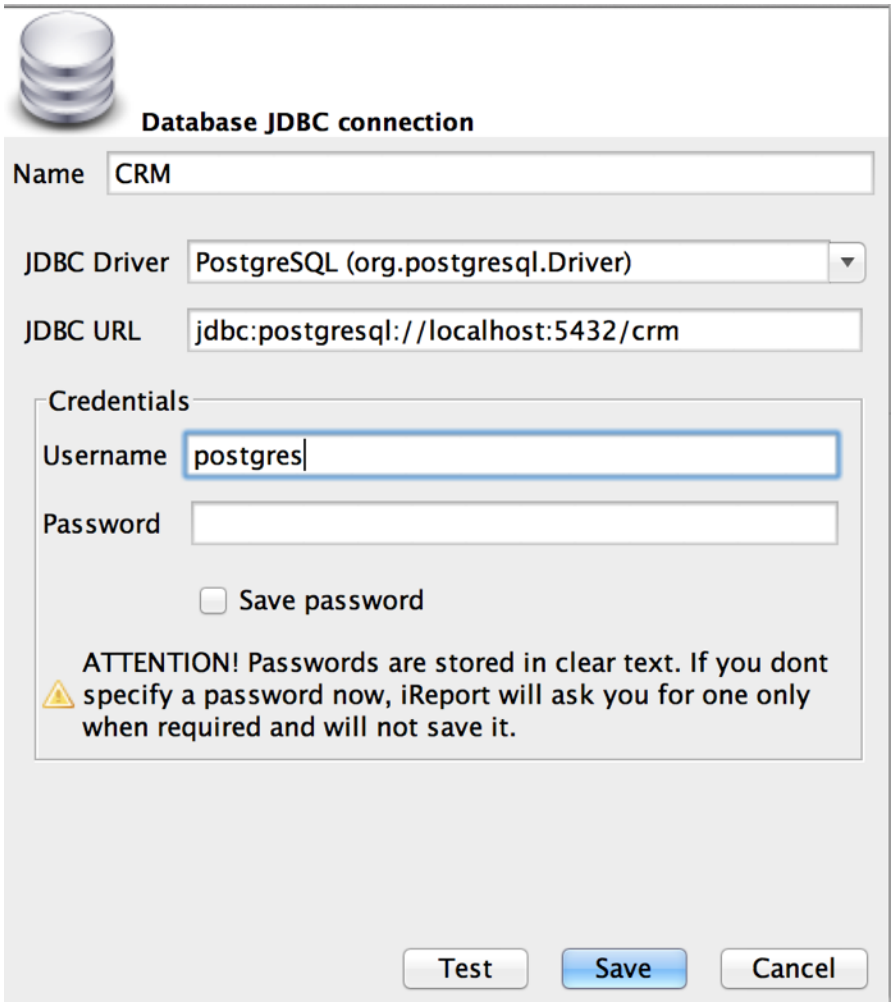


Choose “JDBC Connection” and click “Next”.

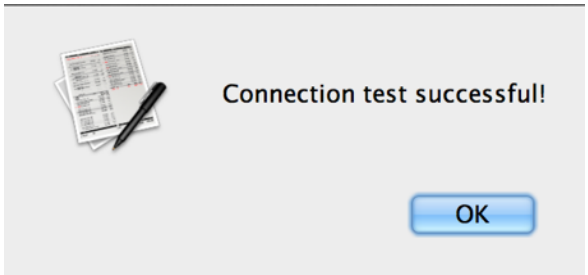
Now fill out the connection properties:

1. Name: An arbitrary name that will appear in the datasource list which you can use to describe these properties
2. JDBC Driver: This setting tells iReport which driver to use for connecting to the CRM database; from the drop-down choose “PostgreSQL (org.postgresql.Driver)” to connect to a PostgreSQL database
3. JDBC URL: This is a specific connection string that must be correctly entered in order to connect to the CRM database; this string is based on the JDBC Driver that was selected

4. Username: The username that your database administrator has provided for you to access the database; read-only access is required
5. Password: The password that matches the database username



Test the connection before saving it. You should see a successful test message:



If successful, save the settings. If unsuccessful, review the properties carefully and check your username and password. iReport will provide some indication of why the connection failed.

You are now ready to design a report.

Report Criteria

Before actually using iReport to layout the report elements, a thorough understanding of the report will speed things along. You might find the following steps helpful:

Step 1: Define the Report

Each report begins with an idea that must be conveyed. You might come up with some sort of simple declaration of what the report should achieve to help with the design. Be specific. This can be a formal statement or an informal list.

You will want to include the type of report: whether a list of items or a detailed view of a specific item; the data you would like to see on the report; how the report will be organized and sorted; and whether there are any user selectable parameters to choose from when the report is executed by an end-user.

Example report definition:

This report will list all the revenue of a specific account, including the amount, organization, and the payment date. The organizations will be grouped alphabetically, then sorted by the payment date. When this report is executed, the user will be allowed to filter the report by selecting a specific account.

Step 2: Understand the Data

In order to generate the report, you will need to know which database tables are required, how those tables relate to each other, and which fields are required to satisfy the report.

An understanding of the application modules and the data contained in the modules will be most helpful. The database schema is a great resource in determining the tables and fields, and will aid in designing the initial SQL query.

Example report data:

Tables required:

[organization]
[revenue]

The data for an “organization list” report comes from the [organization] table. This table must be joined to the [revenue] table to retrieve the organization’s payment data.

A dynamic parameter will need to be applied to the SQL query to restrict the data by account.

The data will need to be sorted by organization, then by the revenue received field.

Step 3: Write a Database Query

Now that the fields, tables, and parameters are known, it's time to write a query that will output the required data. This query can be either entered into iReport (see the next section on Writing a Report) or into a 3rd party SQL tool which allows you to easily test your query statement.

Be sure to carefully verify that the returned data from the query is exactly what is needed for the report. Set any of the parameters in the where clause with hard-coded data as these will be replaced with parameter tags later in iReport.

The following query will return the values required for the "organization list" example:

```
SELECT o.name, o.org_id, r.received, r.amount
FROM organization o
LEFT JOIN revenue r ON (o.org_id = r.org_id)
WHERE o.org_id > -1
AND r.amount IS NOT NULL
AND r.amount > 0
ORDER BY o.name, o.org_id, r.received
```

Report Design

Now that you have an understanding of the report, the various elements can now be defined in iReport.

Start a New Report: Report Wizard

The first step is to create a report in iReport by choosing the "File" menu then selecting either the "Report Wizard" or "New Document" menu item. This tutorial will use the Report Wizard to quickly input the report query, choose the fields for the report, specify the grouped data, and then finally specify the desired layout.

Manually starting a document will provide you with complete control of the report layout process from start to finish. Using the report wizard will assist you in inputting the main criteria, then allowing you to optionally refine the layout later.

Step 1: Query

The first step in the wizard is to input the query to be used. From the "organization list" example, the following query can be used:

```
SELECT o.name, o.org_id, r.received, r.amount
FROM organization o
LEFT JOIN revenue r ON (o.org_id = r.org_id)
WHERE o.org_id > -1
AND r.amount IS NOT NULL
AND r.amount > 0
ORDER BY o.name, o.org_id, r.received
```

Choose "Next >" to continue. If there was an error in testing the query, iReport will provide an error message.

Step 2: Fields Selection

iReport will execute the query and display all of the fields from which you can choose from. Select the fields in the order in which they should appear in the report.

For this report, the fields are selected in this order: name, org_id, received, amount.

Choose "Next >" to continue.

Step 3: Group By...

Since this example report specifies that the list must be grouped by the organization, select "name" from the "Group 1" drop-down.

Choose "Next >" to continue.

Step 4: Layout

This report is intended to display payment details in a list. Choose "Tabular" from the layout type drop-down, then choose a sample layout, like "classic_landscapeT.xml"

Choose "Next >" to continue.

Step 5: Finish

To generate the report layout, simply choose "Finish" and iReport will generate a generic layout based on the fields chosen. At this point the report can be tested and additionally you might want to improve or alter the report.

Test the Report

Once the wizard is finished successfully, you can choose to preview the report. From the Design/XML/Preview tabs, choose the option "Preview."

iReport will now compile the report and display the report with the current active database connection. If there are any compilation errors they will be presented. If no errors, then the report will be executed and displayed in the window.

Report Basics

The visual report layout is made up of bands, static text elements, database field elements, shapes, lines, and variables. All of which can now be modified.

- Bands: report elements exist within report bands; these elements are displayed based on the band definition; common bands include: title, page header, column header, detail, page footer, last page footer, and summary
- Static Text: Report titles and column names, by default, are static text; static text can be modified by double-clicking on the element and then choosing among: Common, Font, Static Text, and Border characteristics

- Fields: a dynamic values are placed in a field element; the text field is displayed as: $F{field_name} where field_name is the name of the field to display; fields can include values retrieved from the database, calculated variables, and even code
- Shapes: Lines, images, boxes, and more can be placed on the report
- Variables: a dynamic value that is generated during report execution; variables can keep track of page numbers, record counts, and more

Report Changes

Now that you have reviewed the report, you might notice it looks a little strange, only because the report labels and layout came from the database. Here are some suggestions that might improve the readability and presentation of the report:

1. Give the report a new title; by default it says "Classic Report Template"
2. Rename the column names from their generated schema name to a more presentable display name; rename the columns to: Account Name, Payment Date, Payment Amount
3. Fields can be resized so that the width of the field represents the data that is contained -- notice how each field is spaced evenly; you might want the account name field to be longer; or else data might be truncated when viewed
4. In some columns, the value might be 'null' so you need to create a report expression to output a blank value instead of the word 'null'

Using Report Parameters

Before the report can be used in the CRM, any hard-coded values need to be changed in the SQL query so that the CRM can either prompt users for a value or so that the CRM can supply values automatically. There are parameters for choosing values from drop-down lists, for limiting results to the user's 'owned' accounts, and more.

From the "View" menu, choose the "Report Parameters" menu item.

In the popup window, a list of built-in parameters is displayed, choose to create a new parameter. The parameter name must match the pre-programmed names exactly used by ConcourseSuite (see the parameter reference for details).

In this example, set the following properties:

- Parameter name (must match exactly): orgid
- Parameter class type: java.lang.Integer
- Is for prompting: Yes (checked)
- Default value expression: new Integer(1)
- Parameter description: Account

Close the window, and modify the report query to use the new parameter by performing the following:

In the Report SQL Query, add a line that reads:

```
WHERE o.org_id = $P{orgid}
```

Choose “OK” to validate and save the updated query.

Now, if you execute the query again, iReport will prompt you for the org_id to use in the query. When this report is deployed to the CRM, the UI will appear with a selector for an account to choose from when the user runs this report.

Installing a Report

Once you are satisfied with the report, a CRM developer can deploy the report in the CRM.

1. Place the report source files in /WEB-INF/reports
2. Update the permissions_en_US.xml file and associate the report with the correct module
3. Package and deploy the application
4. Write a script to upgrade existing CRM installations; update the ApplicationVersion.java file to trigger an update
5. Reports can also just be copied into the appropriate directory and have the database updated

```
/* Install a jasper report script
 * - The .xml will be copied by ant
 * - This script inserts the report into the database so users can use it
 */
import com.concursive.crm.db.DatabaseUtils;
import com.concursive.crm.web.modules.reports.dao.Report;
import com.concursive.crm.web.modules.admin.dao.PermissionCategory;
import com.concursive.crm.web.modules.admin.dao.Permission;

// Using the constant categoryId from permissions.xml, get the database id for this
module
int categoryId = PermissionCategory.lookupId(db, 1); //Accounts
if (categoryId != -1) {
  // Set these and the rest will insert the new report
  String filename = “accounts_revenue.xml";
  String permissionName = "accounts-accounts-revenue";
  int typeId = 1;//not implemented as of this date
  String title = “Account revenue";
  String description = "What are the recent payments by account?";

  // Check to see if report already exists
  int reportId = Report.lookupId(db, filename);
  if (reportId == -1) {
    // Update the category report capability
    PermissionCategory.updateReportAttribute(db, categoryId, true);
    // Get the permission id for accessing this report
    int permissionId = Permission.lookupId(db, permissionName);
    // Insert the report
    Report report = new Report();
    report.setCategoryId(categoryId);
    report.setPermissionId(permissionId);
    report.setFilename(filename);
    report.setType(typeId);
    report.setTitle(title);
    report.setDescription(description);
    report.setEnteredBy(0);
    report.setModifiedBy(0);
    report.insert(db);
    print("Report inserted: " + filename);
  } else {
    print("Report already exists: " + filename);
  }
}
```

Programming Report Parameters

Determine what the parameter is supposed to do

Report parameters have been designed with a pattern in mind. This helps to organize them visually, but also so that they can be programmatically executed without additional coding.

For example, all parameters beginning with "lookup_<table name>" tells the application that a LookupList of values is needed from the <table name> specified. This is a dynamic parameter since no additional programming is required if the report writer needs to retrieve a lookup list for a newly added database table.

Additional parameter models will need to be added as needs arise.

Prepare the dependent parameter information to show the user

In Parameter.java, the prepareContext(HttpServletRequest, Connection) method is responsible for preparing information for display to the user, just before they run the report.

For example, if the user is to choose from a drop-down list to specify a report value, then the drop-down list values must be retrieved from the database (or cache). In Parameter.java there is a generic code block that reads values from the database by using any specified table name.

This method just prepares the data to be used in the presentation to the user, it does not generate the actual HTML that is displayed to the user, that is a separate step.

Generate the HTML for the parameter

All parameters that the user is prompted for are displayed using HTML. In Parameter.java, the getHtml(HttpServletRequest) method is responsible for generating the HTML for a corresponding element. This method is actually called by a JSP while the parameters are being presented to the user, so a database connection is no longer available. Any data that was prepared in the previous step is now available here for display.

For example, in the previous step, a LookupList was generated by querying the database. The LookupList was added to the request and is now available here in the JSP. The getHtml() method simply calls LookupList.getHtmlSelect() which in turn contains code for outputting HTML.

When adding a new parameter, you will need to know what the HTML will need to look like... will it be a text field? A series of combo-boxes? A text field with a Calendar widget? Etc.

Parameter Reference

The CRM software includes several pre-programmed parameters that can be used in reports. Some of the parameters prompt the user for data, some of the parameters are automatically inserted by the application. Look in com.concursive.crm.web.modules.reports.dao.ParameterList.

Report Parameter Reference

Parameter	Prompt User?	HTML	Description	Type
$P{path}	No		Inserts the path in which reports and subreports can be found	java.lang.String
$P{path_icons}	No		Inserts the web application's images/icons path	java.lang.String
$P{user_name}	No		Inserts the user's name	java.lang.String
$P{userid}	No		Inserts the user's id, typically for retrieving this user's data only	java.lang.Integer
$P{userid_range_source}	Yes	Combo box	Asks user for “My Records” or “Hierarchy-Controlled Records”	java.lang.String
$P{userid_range}	No		The comma-separated value returned from $P{userid_range_source}	java.lang.String
$P{text_[name]:255}	Yes	Text field	Displays a text field intended for arbitrary text; optionally with the specified character limit	java.lang.String
$P{number_[name]:5}	Yes	Text field	Displays a text field intended for numbers only; optionally with the specified character limit	java.lang.Integer
$P{date_[name]}	Yes	Text field and calendar	Asks the user for a valid date; [name] can be arbitrarily named	java.sql.Timestamp
$P{lookup_[name]}	Yes	Combo box	Shows a combo box from the lookup table specified and returns the Id that was selected from the lookup table; *see $P{lookup_[name]_where}*	java.lang.Integer

Parameter	Prompt User?	HTML	Description	Type
$P{lookup_[name}_where}	No		If $P{lookup_[name]} returns a number greater than -1, then the description of this parameter is substituted in the query; for example, the description of this parameter could be: AND x = $P{lookup_[name]} The query would specify: $P!{lookup_[name]_where}	java.lang.String
$P{orgid}	Yes	Select an organization widget	Prompts user to choose an organization from a popup; user does not have to make a choice	java.lang.Integer
$P{orgid_where}	No		If $P{orgid} returns a number greater than -1, then the description of this parameter is substituted in the query	java.lang.String
$P{percent_[name]}	Yes	Combo box	A combo box of probabilities will be shown to the user, for example: “All”, “> 10%”, ..., “< 90%” ...; *see $P{percent_[name]_min} and $P{percent_[name]_max}*	java.lang.Integer
$P{percent_[name]_min}	No		The minimum amount of the actual percent range the user chose is stored and can be used in a query, for example, the value might be: WHERE x > $P{percent_[name]_min}	java.lang.Double
$P{percent_[name]_max}	No		The maximum amount of the actual percent range the user chose is stored and can be used in a query, for example, the value might be: WHERE x < $P{percent_[name]_max}	java.lang.Double
$P{lookup_ticket_priority}	Yes	Combo box	Displays the available ticket priorities to choose from; returns the Id selected; *see $P{ticket_priority_where}*	java.lang.Integer
$P{lookup_ticket_priority_where}	No		*Similar to $P{lookup_[name]_where}*	*java.lang.String*
$P{lookup_ticket_severity}	Yes	Combo box	Displays the available ticket severities to choose from; returns the Id selected; *see $P{ticket_severity_where}*	java.lang.Integer
$P{lookup_ticket_severity_where}	No		*Similar to $P{lookup_[name]_where}*	*java.lang.String*

Parameter	Prompt User?	HTML	Description	Type
$P{lookup_state}	Yes	Combo box	Displays the states/provinces to choose from, the selected abbreviation is stored	java.lang.String
$P{lookup_state_where}	No		*Similar to $P{lookup_[name]_where}*	*java.lang.String*
$P{lookup_call_result}	Yes	Combo box	Displays a list of call results to choose from	java.lang.Integer
$P{lookup_call_result_where}	No		*Similar to $P{lookup_[name]_where}*	*java.lang.String*

Example XML usage:

<parameter name="CENTRIC_DICTIONARY" isForPrompting="false" class="java.util.Map"/>
<parameter name="SCRIPT_DB_CONNECTION" isForPrompting="false"
class="java.sql.Connection"/>
<parameter name="INSTANCE_ID" isForPrompting="false" class="java.lang.Integer">
<parameter name="path" isForPrompting="false" class="java.lang.String">
<parameter name="currency" isForPrompting="false" class="java.lang.String">
<parameter name="country" isForPrompting="false" class="java.lang.String">
<parameter name="language" isForPrompting="false" class="java.lang.String">
<parameter name="userid" isForPrompting="false" class="java.lang.Integer">
<parameter name="user_hierarchy" isForPrompting="false" class="java.lang.String">
<parameter name="user_contact_name" isForPrompting="false" class="java.lang.String">
<parameter name="my_company_orgid" isForPrompting="false" class="java.lang.String">

Example interactive usage:

<parameter name="siteid" isForPrompting="true" class="java.lang.Integer">
<parameter name="lookup_account_types" isForPrompting="true" class="java.lang.Integer">
<parameter name="date_start" isForPrompting="true" class="java.sql.Timestamp">
<parameter name="date_end" isForPrompting="true" class="java.sql.Timestamp">
<parameter name="orgid" isForPrompting="true" class="java.lang.Integer">
<parameter name="userid_range_source" isForPrompting="true" class="java.lang.String">
<parameter name="lookup_role" isForPrompting="true" class="java.lang.Integer">
<parameter name="lookup_department" isForPrompting="true" class="java.lang.Integer">
<parameter name="percent_*" isForPrompting="true" class="java.lang.String">
<parameter name="lookup_stage" isForPrompting="true" class="java.lang.Integer">
<parameter name="boolean_*" isForPrompting="true" class="java.lang.Integer">