Aras Innovator XSLT Report Tool Users Guide

Aras Innovator 9.4

Document #: 9.4.006022011

Last Modified: *7/31/2013*

ARAS CORPORATION

Aras Corporation
300 Brickstone Square
Suite 700
Andover, MA 01810

Phone: 978-691-8900
Fax: 978-794-9826

E-mail: Support@aras.com
Website: http://www.aras.com

Notice of Rights

Notice of Liability

The information contained in this document is distributed on an "As Is" basis, without warranty of any kind, express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose or a warranty of non-infringement. Aras shall have no liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this document or by the software or hardware products described herein.

Table of Contents

Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- **Did you find any errors?**
- **Is the information clearly presented?**
- **Do you need more information? If so, where and what level of detail?**
- **Are the examples correct? Do you need more examples?**
- **What features did you like most?**

If you find any errors or have any other suggestions for improvement, please indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

- **Email:**
  Support@aras.com
  Subject: Aras Innovator Documentation

Or,

- **Postal service:**
  Aras Corporation
  300 Brickstone Square
  Suite 700
  Andover, MA 01810
  Attention: Aras Innovator Documentation

Or,

- **FAX:**
  978-794-9826
  Attn: Aras Innovator Documentation

If you would like a reply, please provide your name, email address, address, and telephone number.

If you have usage issues with the software, please visit http://www.aras.com/support/

1 Introduction

The *Report Tool* is the user interface for creating and maintaining Reports about items and their configurations. The intent of the Report Tool is to offer a very simple user interface for designing Reports.

In technical terms the Reports are the result of an XSLT transformation of an AML item configuration, which is the result of a query. The Report Tool allows direct editing of the XSLT stylesheet.

Reports are relationships on ItemTypes. The Report item has properties that define how to query data for the Report and format it for display. Reports tend to fall into the following types:

1. A report about an item and its relationships. For example, a BOM report. These are *Item* type Reports.
2. A static query report about items by ItemType. For example, an Open ECO report. These are *ItemType* type Reports.
3. A dynamic query report about items by ItemType. The user will interactively enter query criteria for the report. These are *Generic* type Reports.

2 Running Reports

The main menu bar has a Reports menu choice, which is context sensitive as you navigate the main tree. The drop down menu is populated with the Report items related to the ItemType you selected from the tree.

Figure: The main menu bar.

How Reports appear on the Reports menu:

- The *Generic* type Reports are always shown. (Generic Reports will tend to be interactive and the user interface will be implemented in future release.)
- The *ItemType* type Reports are shown when the ItemType is selected from the main tree.
- The *Item* type Reports are only shown when an item is selected. Plus Item Reports appear on the right-mouse context popup menu.

3 Report ItemType

Through the Report Tool you edit Report items, which are instances of the Report ItemType. This section is a brief overview of the Report ItemType and provides a technical understanding of how Reports are implemented in Aras Innovator.

Reports are the result of an AML query or a Method. The Report ItemType implements the properties for you to define the query and XSLT stylesheet or Method. In addition the Report describes where and how the Report is viewed on the client.

The Report ItemType has the following properties:

Property	Data Type	Length/Source	Content/Comments
name	string	32	The name for the Report.
description	string	128	The description for the Report.
type	list	Report Types	This specifics the type of Report: Item, ItemType, or Generic; which defines how the Report is handled by the server and when it appears on the Reports main menu bar.
target	list	Report Targets	This specifies where to show the Report results: in the main window, its own window, or all reports in one window when many items are multi-selected.
label	Multilingual String	32	This allows us to specify the text, in different languages, of Menu entry for this Report under the Reports Menu of the Client interface.
report_query	text		The AML query to generate data for the Report or an XSLT stylesheet to generate the actual AML query if the Report type is Item.
xsl_stylesheet	text		The XSLT stylesheet used to transform the query results into the Report.
method	item	Method	The Method is evaluated and the result is the Report. If a Report is based on the results of a Method the report_query and xsl_stylesheet property are not used.
location	list	Report Locations	This defines where the logic for generating the Report is performed on the client or server. Typically this is server side but it is possible to run a Method client side to generate a Report.

ItemTypes have relationships to Reports called the "Item Report" RelationshipType. You can create new Reports and edit existing Reports from the Reports tab on the ItemType window. Reports are related items because Generic Reports are not specific to any particular ItemType thus are standalone Reports. Reports tend to be based on an item or items by ItemType and their configurations.

4 Report Tab

When you open a Report item from the Reports tab on the ItemType window the Report Tool will open. This is the standard Tear off Item Window including the standard menu bar and toolbar. Below that is the Report tab bar allowing you to switch between the Report details and the XSLT stylesheet text editor. This section describes the Report tab, which is the Report Form. The Report tab is the page where you setup the Report. Here you provide its name, its type, the target window to show the Report results plus setup the query criteria to generate the data for the Report.

The diagram below is a sample Report that is of type *ItemType*, meaning the Report Query is a static query that will search for Vendor items. The AML query also limits the results by defining only the properties desired using the select attributes. And the target is set to Window so the Report will appear in a new separate window when run from the client.

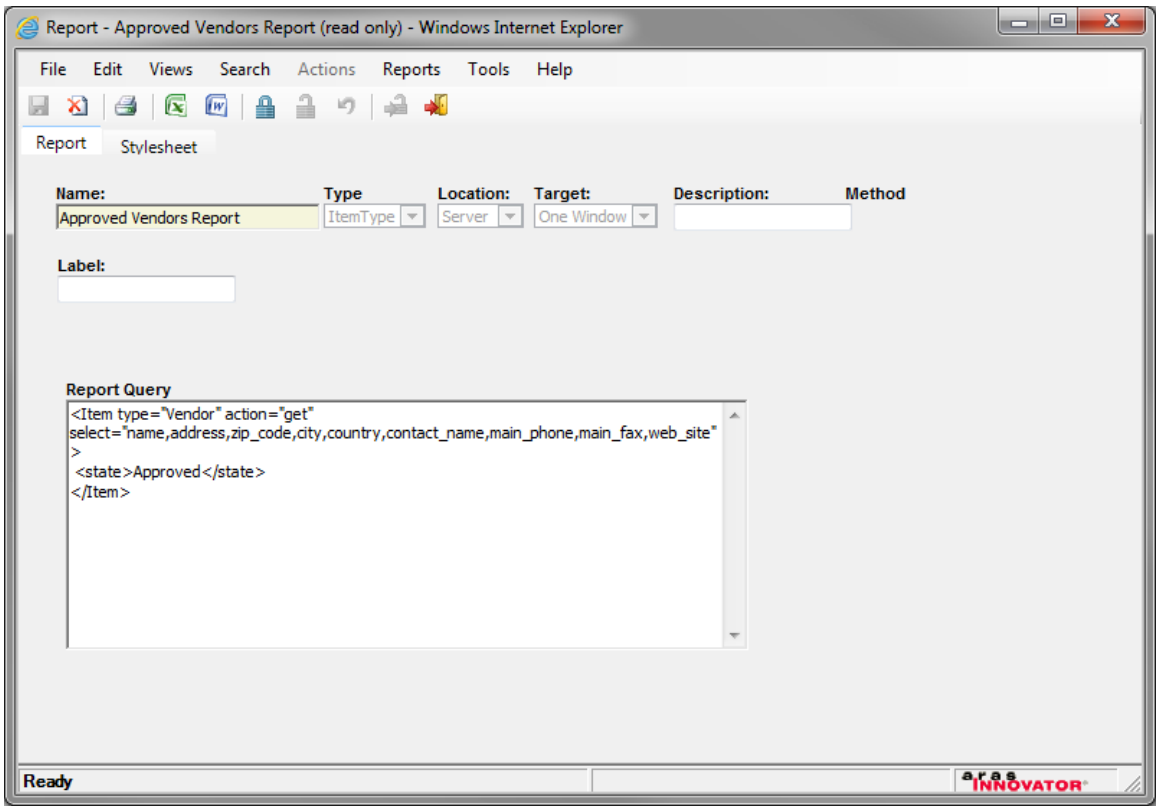

Figure: The Report Tab.

The table below provides a description for each field on the Report Form.

<table>
<tr><th>Field</th><th>Content/Comments</th></tr>
<tr><td>Name</td><td>The name for the Report.</td></tr>
<tr><td>Description</td><td>The description for the Report.</td></tr>
<tr><td>Type</td><td>
<table>
<tr><th>Type</th><th>On Menu Bar</th><th>Report Query Contains</th></tr>
<tr><td>Generic</td><td>Always</td><td>The Report Query is an AML query for the Report, but if not set then use the <Item> passed in the body of the client request as the AML query. These are dynamic Reports where the query is known at run time.</td></tr>
<tr><td>ItemType</td><td>When selected off the main tree.</td><td>The report_query is the AML query.</td></tr>
<tr><td>Item</td><td>When selected off the main search grid.</td><td>The report_query is an XSLT stylesheet applied against the <Item> passed in the body of the client request as the AML query, which is the selected item from the main search page.</td></tr>
</table>
</td></tr>
<tr><td>Target</td><td>Where to show the Report results: in the main window, its own window, and all reports in one window.</td></tr>
<tr><td>Location</td><td>This defines where the method for generating the Report is evaluated on the client or server.</td></tr>
<tr><td>Report Query</td><td>The AML query or XSLT stylesheet to make the AML query from the item depending on if the type is set to Item or ItemType.</td></tr>
<tr><td>Label</td><td>This allows us to specify the text, in different languages, of Menu entry for this Report under the Reports Menu of the Client interface.</td></tr>
</table>

4.1 Report Query

When the Report type is set to ItemType or Generic the Report Query is the static AML query to generate the data for the Report. There are a few attributes that are very useful to help filter and control the results for your Report.

select – the select attribute is used to define the set of properties you want returned from the query. This is a comma delimited list of property names (not the property label). This is identical to the select clause in SQL if you are familiar with relational database technology. You can gain significant performance improvements to your Reports by limiting the data returned from the query.

The following example illustrates the select attribute by selecting the item_number, description, and cost properties from the Part ItemType.

```
<Item action="get" type="Part" select="item_number,description,cost"/>
```

order_by – the order_by attribute is used to order the results. The Report Tool offers the ability to sort the data in a variety of ways but sometimes you may want the data in a pre-sorted order. This is a comma delimited list of property names (not the property label). This is identical to the order by clause in SQL if you are familiar with relational database technology.

The following example illustrates the select attribute by selecting the item_number, description, and cost properties from the Part ItemType.

```
<Item
 action="get"
 type="Part"
 select="item_number,description,cost"
 order_by="item_number"/>
```

page & pagesize – if you are performing a query for items and you know that there can be hundreds or thousands of items you can limit the number of items returned by using the page and pagesize attributes. Together they control how many items to return (the pagesize) and which page to return.

The following example illustrates the select attribute by selecting the item_number, description, and cost properties from the Part ItemType.

```
<Item
 action="get"
 type="Part"
 select="item_number,description,cost"
 order_by="item_number"
 page="1"
 pagesize="50"/>
```

Relationships – often you will want the relationship items in addition to the source item to generate a Report showing the items configuration. The query may contain the <Relationships> tag to hold the <Item> tags for the relationship items you want included in the query results. The <Item> tags can themselves include the same control attributes.

The following example illustrates the Relationships tag to include the BOM relationships. Notice the select attribute for the BOM type Item defines the properties to be returned for both the BOM relationship item plus the properties for the Part item referenced by the related_id property, which is defined within the parenthesis following the related_id property.

```
<Item
  action="get"
  type="Part"
  select="item_number,description,cost"
  order_by="item_number">
  <Relationships>
    <Item
      action="get"
      type="Part BOM"
      select="qty,ref_des,related_id(item_number,description,cost)"/>
  </Relationships>
</Item>
```

4.2 Stylesheet Query

When the Report type is set to *Item* the Report Query is an XSLT stylesheet, which is used to transform AML for the selected item into a new AML query that is then used to query the data for the Report. The reason is the item selected tend not to already include the relationships for the item so you need to generate the actual query dynamically using the data known from the selected item.

For example, the following is an XSLT stylesheet that transforms a Part item into a new AML query for a BOM Report. Notice that for the most part this is actually just an AML query but because it's an XSLT stylesheet the *Attribute value templates* `{…}` can be used to substitute values inline from the selected item. The text within the curly brackets is evaluated as an expression, which in this case is simply returning the values for the type and id attributes from the selected item.

```
<Item
  action="get"
  type="{@type}"
  id="{@id}"
  select="item_number,description,cost">
  <Relationships>
    <Item
      action="get"
      type="Part BOM"
      select="qty,ref_des,related_id(item_number,description,cost)"/>
  </Relationships>
</Item>
```

5 Stylesheet Tab

The Stylesheet Tab presents the XSLT stylesheet for the Report in its native format so you can edit the stylesheet to make changes. This is a simple text area to edit the XSLT stylesheet directly.

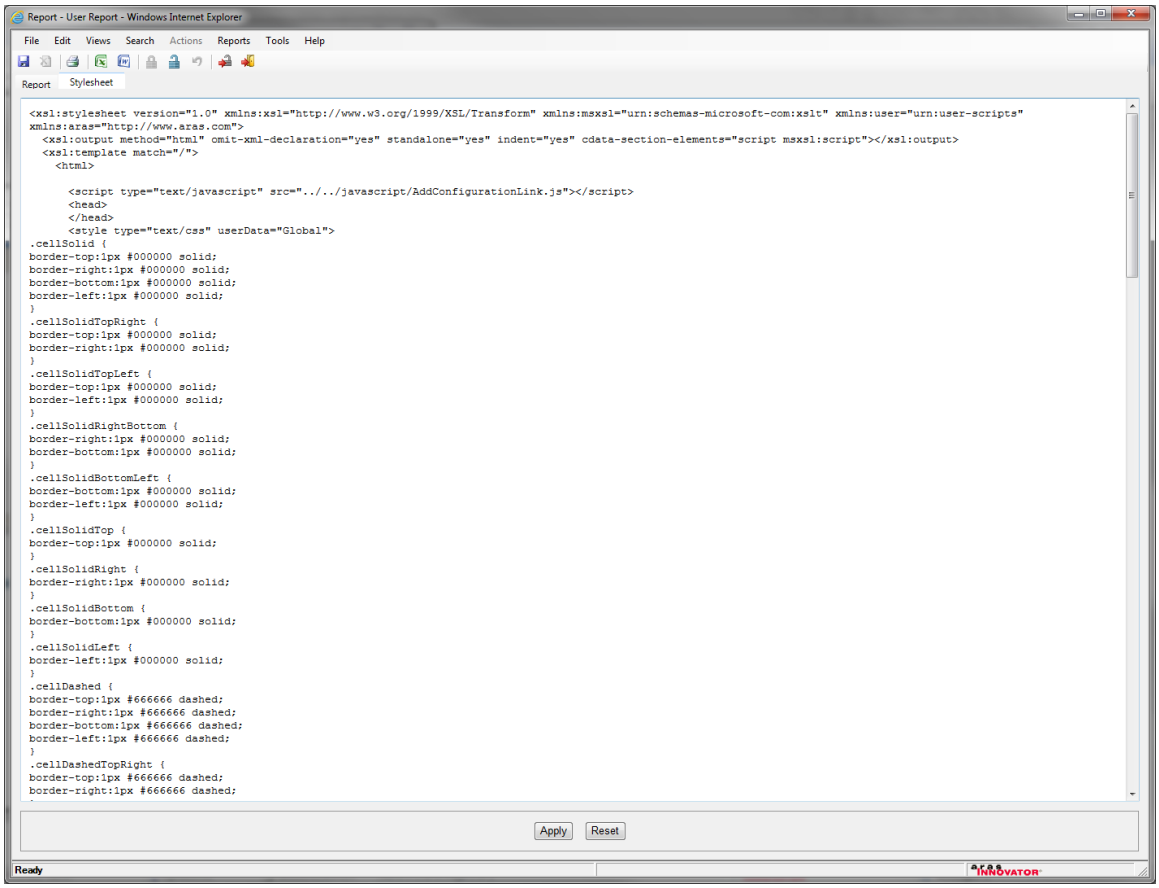

Figure: The Stylesheet Tab.

- The Apply button will apply the changes.
- The Reset button will reset the text back before the changes.

6 Report Tool Tips and Tricks

This document is a guide to help you become more productive with the Report Tool by showing you how to address some of the more common tasks that need to be done when designing a Report.

6.1 Query Tips

1. *Using the select attribute.*

   The select attribute is how you can filter down the set of properties for the Items returned from a query.

```
<Item type="Part" action="get" select="part_number,part_description,part_cost"/>
```

2. *Using the order_by attribute.*

   The order_by attribute is how you can order the results of the query. The stylesheet also supports sorting on the properties however.

```
<Item type="Part"
      action="get"
      select="part_number,part_description,part_cost"
      order_by="part_number"
/>
```

3. *Using the condition attribute.*

   The condition attribute is how you can specify a condition for the properties used as search criteria. All the conditions that are allowed in SQL are allowed as the values to the condition attribute. Note however, that the greater then and less then symbols are replaced with the two letter pneumonic word. The < symbol is *lt* and the > symbol is *gt*, >= is *ge* and so on.

```
<Item type="Part"
      action="get"
      select="part_number,part_description,part_cost"
      order_by="part_number">
  <part_number condition="like">%123%</part_number>
</Item>
```

4. *OR'ing property values.*

   The <or> tag provides a way too logically OR property values for the search criteria.

```
<Item type="Part"
      action="get"
      select="part_number,part_description,part_cost"
      order_by="part_number">
  <or>
    <state>Released</state>
    <state>In Review</state>
  </or>
</Item>
```

5. *Return Relationships in the results.*

   You can include the <Relationships> tag and the relationship items you want to return.

```
<Item type="{@type}" id="{@id}" action="get" select="part_no,part_desc,part_cost">
  <Relationships>
    <Item type="Part EBOM" action="get" select="position,qty,related_id">
      <related_id>
        <Item type="Part" action="get" select="id,part_no,part_desc,part_cost">
          <Relationships>
            <Item type="Part AVL" action="get"
select="avl_status,related_id(description,manufacturer,component_no)"/>
          </Relationships>
        </Item>
      </related_id>
    </Item>
  </Relationships>
</Item>
```

6. *Using Relationships as search criteria.*

   You can include the <Relationships> tag to describe the search criteria for the relationship items.

```
<Item type="{@type}" id="{@id}" action="get" select="part_no,part_desc,part_cost">
  <Relationships>
    <Item type="Part EBOM" action="get" select="position,qty,related_id">
      <qry condition="gt">1000</qty>
    </Item>
  </Relationships>
</Item>
```

6.2 Stylesheet Tips

1. *Displaying a non-breaking white space.*

   To display a non-breaking white space you can use the <xsl:text> tag.

```
<xsl:text disable-output-escaping="yes">&amp;nbsp;</xsl:text>

Or this:  
```

2. *Displaying the formatted text property.*

   To display formatted text (the marked up text) property value in XSLT you need to use the disable-output-escaping attribute with the <xsl:value-of> tag. Plus the marked up text for the formatted text property is a complete HTML document, which you only want the content for the BODY tag.

```
<xsl:value-of disable-output-escaping="yes" select=".//BODY/*"/>
```

3. *Displaying nested tables for relationships.*

   To display the relationship Items for nested tables you will need to reduce the XPath to the relative path based on the context node for the <xsl:for-each> loop that the nested relationships table is in.

```
<xsl:for-each select="Relationships/Item">
  <xsl:for-each select="related_id/Item/Relationships/Item">
  </xsl:for-each>
</xsl:for-each>
```

4. *Resize the Report window.*

   You want to resize the window that Report is shown in. You need to add an onload callback function for the HTML page. Add the following code before the <body> tag.

```
<script>
onload = function()
{
  top.window.resizeTo(800,600);
}
</script>
```

5. *Show a different value based on the value for a property.*

   You want to show a different value in the Report based on the actual value of the property. For example, the actual value of the property is the hex number for a color but you want to show the text word for that color.

```
<xsl:choose>
  <xsl:when test="mgr_opinion='#FF0000'">Red</xsl:when>
  <xsl:when test="mgr_opinion='#FFFF00'">Yellow</xsl:when>
  <xsl:when test="mgr_opinion='#00FF00'">Green</xsl:when>
</xsl:choose>
```

6. *Using script functions in the stylesheet.*

   You want to call a JavaScript or VBscript function in the stylesheet. There is a predefined namespace named "user" that you can use to implement your <msxsl:script> block. The first step is to add the JavaScript or VBscript to your stylesheet and the second step is to call the function defined. The following example shows how to call the VBscript DatDiff() function.

```
<msxsl:script language="VBScript" implements-prefix="user">
  Function myDateDiff(day)
    myDateDiff = DateDiff("d", day.item(0).text, Now)
  End Function
</msxsl:script>
```

   You would add the code above in the stylesheet after the <xsl:stylsheet> tag. Then you can call the function using the <xsl:value-of> tag like this:

```
<xsl:value-of select="user:myDateDiff(created_on)"/>
```

7. *Supporting European languages.*

   You want to show the results in a European language.

   1) Must set the encoding attribute for the root xml tag:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

   2) Must set the encoding attribute for the output tag:

```
<xsl:output encoding="iso-8859-1" method="html"
omit-xml-declaration="yes" standalone="yes" indent="yes" />
```

7 AML Schema

The following is the AML Schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="Item">
    <xsd:all>
<!– Any user elements defined by the ItemType for the item →
      <xsd:complexType name="Relationships" minOccurs="0"
maxOccurs="1">
        <xsd:all>
          <xsd:element name="Item" minOccurs="0"
maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:all>
<!– Any user elements defined by the ItemType for the item →
                <xsd:attribute name="id" type="xsd:ID" />
                <xsd:attribute name="type" type="xsd:string" />
                <xsd:attribute name="typeID" type="xsd:IDREF" />
                <xsd:attribute name="action" type="xsd:string"
/>
                <xsd:attribute name="sort_order"
type="xsd:integer" />
              </xsd:all>
            </xsd:complexType>
          </xsd:element>
        </xsd:all>
      </xsd:complexType>
    </xsd:all>
    <xsd:attribute name="id" type="xsd:ID" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="typeID" type="xsd:IDREF" />
    <xsd:attribute name="action" type="xsd:string" />
    <xsd:attribute name="select" type="xsd:string" />
    <xsd:attribute name="order_by" type="xsd:integer" />
    <xsd:union>
      <xsd:attributeGroup>
        <xsd:attribute name="pagesize" type="xsd:integer" />
        <xsd:attribute name="page" type="xsd:integer" />
      </xsd:attributeGroup>
      <xsd:attributeGroup>
        <xsd:attribute name="levels" type="xsd:integer" />
        <xsd:attribute name="config_path" type="xsd:string" />
      </xsd:attributeGroup>
    </xsd:union>
  </xsd:complexType>
</xsd:schema>
```

8 Cookbook

This section provides a set of recipes for doing common reporting tasks.

8.1 User Input for Reporting Services based Reports

You need a Form to collect user input for Named Parameters for the SQL query for Reporting Services based Reports.

Technique
An XSLT based Report is used to present a Form to collect the user input. The user input is mapped to a query_string for the Report Server request via the RSGatwway.aspx page.

In the \Innovator\Client\scripts\ReportTool\ folder there is a UserInputTemplate.txt file that you can use as a template to build the XSLT based Form. This provides a generic way to define the Fields for the user input Form, which opens a modal dialog with a generic HTML page dynamically building the Form based on the Fields defined in the template file.

Procedure for building a user input Form for Reporting Services based Reports with Named Parameters:

1. Login to Aras Innovator as admin.
2. Navigate to the Administration->Reports off the Main Tree.
3. Open a new Report Item.
4. Set the Report Item properties as following:
    a. Name = the name of your Report as deployed in Visual Studio to the Report Server.
    b. Type = Generic or ItemType depending on when you want the Report to appear on the Report menu; Generic always, ItemType only when that ItemType is selected off the Main Tree.
    c. Location = Server; this is required because the XSLT page will return an HTML page with JavaScript that needs to be parsed and Client side Report only writes the transformed page to the browser window and thus the JavaScript is not parsed.
    d. Target = Window; again because the page has JavaScript we need a window to parse the JavaScript. This page will present status info to the user as the Report is being processed and the final Report results will be shown in this same window.
    e. Report Query = leave blank it is currently not used for this purpose.
5. Click on the Stylesheet Tab to insert the template code for setting up the Fields for the modal dialog:
    a. Add the following line under the <HTML> tag

```
<script type="text/javascript"
src="../../javascript/AddConfigurationLink.js"></script>
```

b. Scroll to the bottom of the page.

c. Open the ReportTool/Templates/UserInputTemplate.txt file and select the whole document content.

d. Replace the <body> tag in the Stylesheet Tab with the content from the UserInputTemplate.txt file.

e. Edit the userInputFields object, which is an IOM Item by setting the properties for the Item to match the Named Paramaters for the SQL query for the Report as deployed to the Report Server.

```
///
/// Set the User Input Fields...
/// The property names must match exactly the names you used for the
/// Named Parameters in the SQL Query.
///
/// For example, this SQL Query for the GetParts Report in the
/// Report Server has the following
/// Named Parameters: part_number, description, and cost;
///
/// SELECT     part_number, description, cost
/// FROM         MY_PART
/// WHERE     (part_number LIKE @part_number)
///      AND    (description LIKE @description)
///      AND    (cost LIKE @cost)
///
var userInputFields = opener.top.myInnovator.newItem();
userInputFields.setProperty('part_number', '%');
userInputFields.setProperty('description', '%');
userInputFields.setProperty('cost', '%');
```

f. Click the Apply button at the bottom of the Stylesheet Tab.

i. Please note that the following lines can cause issues when editing this Report XSLT. Ensure that the variables are not commented out before applying and saving.

```
   // Set the URL to the RSGateway passing the user input from the Form
dialog as the query_string.
    var url   = opener.top.aras.getServerBaseURL() +
"RSGateway.aspx?irs:Report=" + reportName;
    var nodes = userInputFields.node.selectNodes('*');
```

g. Click the Save button on the Report Tool toolbar.

That's it, the Report Item should now appear on the Report menu and when selected the User Input Dialog should open. When the user clicks the Submit button the user input is mapped as a query_string and the actual Report is requested from the Report Server via the RSGateway.

If you want to add validation or rearrange the Fields on the Form you can make a copy of the ReportTool/UserInputDialog.html page and modify it to meet your needs.

If you have the need to pass attributes or property values from the selected context Item use the following technique to set the values to hidden form fields:

1. Edit the section from the UserInputTemplate.txt file to pack the desired Item attributes and properties into hidden form fields:

```
<!--
  The following is an example for passing attributes and properties
  from the selected context Item as hidden fields.

  For example, pack the id attribute from the context Item using
  XSLT Attribute Substitution and nested xsl tags for the properties.

  <form>
    <input type="hidden" name="id" value="{//Item/@id}/>
    <input type="hidden" name="name">
      <xsl:attribute name="value" select="//Item/name"/>
    </input>
  </form>
  -->
```

2. Also edit the section from the UserInputTemplate.txt file to pass those hidden field values as Field values for the user input dialog:

```
///
/// If there are properties from the selected context Item packed
/// as hidden fields on the form use this:
///
userInputFields.setProperty('id', document.form[0]['id'].value);
userInputFields.setProperty('name', document.form[0]['name'].value);
```

8.2 Report Chooser

You need a dialog to present a set of Reporting Services based Reports to choose from for the selected Item.

Technique
An XSLT based Report is used to present a dialog to get the Report choice. The user selection is mapped to a query_string for the Report Server request via the RSGatwway.aspx page. This is similar to recipe 8.1 User Input Form but in this case we do want the ID for the selected Item to be passed as a Named Parameter on the query_string.

In the ReportTool/Templates folder there is a file ChooseReportTemplate.txt that you can use as a template to build the XSLT based Form. This template provides a generic way to define the Reports choices and it opens a modal dialog.

Procedure for building a Report Chooser dialog for Reporting Services based Reports:

1. Login to Aras Innovator as admin.
2. Navigate to the Administration->Reports off the Main Tree.
3. Open a new Report Item.
4. Set the Report Item properties as following:
    a. Add the following line under the <HTML> tag

    ```
    <script type="text/javascript"
    src="../../javascript/AddConfigurationLink.js"></script>
    ```

    b. Name = the name of your Report as deployed in Visual Studio to the Report Server.
    c. Type = Item (You do not need the report_query template when prompted).
    d. Location = Server; this is required because the XSLT page will return an HTML page with JavaScript that needs to be parsed and Client side Report only writes the transformed page to the browser window and thus the JavaScript is not parsed.
    e. Target = Window; again because the page has JavaScript we need a window to parse the JavaScript. This page will present status info to the user as the Report is being processed and the final Report results will be shown in this same window.
    f. Report Query = leave blank it is currently not used for this purpose.
5. Click on the Stylesheet Tab to insert the template code for setting up the Report choices for the modal dialog:
    a. Scroll to the bottom of the page.
    b. Open the ReportTool/Templates/ChooseReportTemplate.txt file and select the whole document content.
    c. Replace the <body> tag in the Stylesheet Tab with the content from the UserInputTemplate.txt file.
    d. Edit the AddReport() calls to the Report choices:

```
///
/// Add your Report Choices here...
///
AddReport('Report1', 'Report One', 'rs:Format=Excell');
AddReport('Report2', 'Report Two', 'rs:Format=HTML4.0');
AddReport('Report3', 'Report Three', 'rs:Format=PDF');
```

e. Click the Apply button at the bottom of the Stylesheet Tab.

6. Click the Save button on the Report Tool toolbar.
7. Navigate to the Administration->ItemTypes off the Main Tree.
8. Search for the ItemType for this Report.
9. Edit the ItemType and select the Reports Tab.
10. Add an Existing related Item and select this Report.
11. Click the Save button on the ItemType toolbar.

That's it the Report should now appear on the Report menu when the Item is selected. When the user clicks the Submit button the selected Report the actual Report is requested from the Report Server via the RSGateway.

The ID for the selected Item is automatically passed as a Named Parameter np:id so you must name your Named Parameter in the SQL query with lowercase id also.

8.3 User Input for XSLT based Reports

You need user input for XSLT based Reports.

Technique
Use a Generic Report called from a client Method so that user input can be collect for the query.

This will describe the steps to build the User Input Report. This is just an example and there are alternative ways to implement the user interface but this does illustrate a way to dynamically run a Report.

In this example we will use a very simple Report to get one input from the user, which will be the Part Number for the item_number property for the Part ItemType to dynamically build the AML query for our Report.

The following are the steps for building an Interactive Report:

1. Create a Generic Report as a template.
   a) Set the Report properties as follows:

      Name = My Report (this can be any name you want)
      Type = Generic
      Location = Server
      Target = Window

   b) Construct an AML query and write the Report as usual.

   c) Once the Report is written the AML query is no longer required because the Report will act as a template and the AML query will be provided from the user input interactively so remove the Report Query AML from the Report item. But you may want to use the AML query string as the starting point for building the query in the Method coming up next.

2. Create a client side Method (named "My Report" for this example) that will.
   a. Get the user input for the query, which will be a modal dialog for this example.
   b. Construct the actual AML query based on the user input. Using the AML query from the Report you just did as the starting point. We will replace hard coded search criteria in the AML with variables in our Method.
   c. Create an XML document to hold the AML query.
   d. Get the Report item.
   e. Run the Report passing the query XML document as the query criteria.

3. Create an Action to call the Method. Set the Action properties as follows:

   Type = Generic
   Location = Client
   Target = None
   Method = My Report
   Name = My Report

This provides the logic to get user input and to invoke the Report from an Action menu choice. The following is client side Method code to run the Interactive Report:

```
// The Report is ran as a Generic Report so we can pass our own AML query for the Report.
// Get the query criteria from the user via a modal dialog.
var qryParams = showModalDialog('ReportTool/UserDialogs/MyReport.html', null,
'dialogHeight:227px; dialogWidth:350px; status:0; help:0; resizable:1');
if (! qryParams) return;

with (top)
{
// Create a query Item to hold the AML query for the Report.
// The ItemType Part is an example and you can set this as required.
  var qry = new Item('My Part','get');

// The user input from the modal dialog is serialized the into a string.
// The format for the serialized string is | delimited for the list of properties
// for constructing the query and each property is a name/value pair : delimited.
  var params = qryParams.split('|');
  for (var i=0; i<params.length; ++i)
  {
    var param = params[i].split(':');
    qry.setProperty(param[0], param[1]);
  }

// Get the Report item by name.
  var reportQry = new Item('Report','get');
  reportQry.setProperty('name', 'My Report')
  reportQry.setAttribute('select',
    'name,description,report_query,target,type,xsl_stylesheet,location,' +
    'method(name,method_type,method_code,runas_user,runas_pwd)');
  var report = reportQry.apply();

// Run the report passing the query Item as the query criteria for the Report.
// The runReport function is part of the client Toolkit API,
// which expects node object not IOM Item objects.
  aras.runReport(report.node, '', qry.node);
}
```

4. Create the HTML page that will capture the users input for the Report. In the sample Method code above you will notice the call to open the modal dialog and load it with the ReportDialogs/MyReport.html page. This is the relative URL to the file from the Client/scripts folder for which the Methods on the server side are invoked from.
    a) If the folder Client/scripts/ReportTool/UserDialogs does not exists create it. The folder name can be anything you want; basically we are simply creating a location to put the HTML files for the Report dialogs.

The following is the sample HTML for the MyReport.html page:

```
<html>
  <style type="text/css">
    body {
      background-color:#d4d0c8;
    }
    .header {
      background-color:#CCCCFF;
      font-family:helvetica;
      font-weight:bold;
      font-size:16pt;
      font-style: italic;
      color:#000000;
      border-width:1px;
      border-style:inset;
      padding:8px;
    }
    .buttons {
      background-color:#d4d0c8;
      border-width:2px;
      border-style:groove;
      padding:10px;
    }
    td {
      font-family:helvetica;
      font-weight:bold;
      font-size:8pt;
    }
  </style>

  <script>
  function cancel()
  {
    returnValue = '';
    window.close();
  }

  function apply(form)
  {
    returnValue = '';

// Serialize the field values into a string.
// Delimit the fields with the | character
// and delimit the property name/value with the : character.
// i.e. prop-name:value|prop-name:value

    for (var i=0; i<form.elements.length; ++i)
    {
      var field = form.elements[i];
      if (field.type == 'text')
      {
        if (field.value)
        {
          if (returnValue) returnValue += '|';
          returnValue += field.name + ':' + field.value
        }
      }
```

```
    }
    window.close();
  }

  onload = function() {
    document.body.scroll = 'no';
  }
  </script>

  <body>
    <form>
      <table border="0" cellspacing="0" cellpadding="0" width="100%">
        <tr height="60">
          <td class="header" colspan="2">
            <img src="../../images/icons/32x32/32x32_charts.gif" hspace="20">
            My Report
          </td>
        </tr>
        <tr>
          <td align="right" style="padding:20px 10px 0px 0px;" nowrap>
            Description:
          </td>
          <td style="padding:20px 00px 0px 0px;"><input type="text"
name="part_desc"></td>
        </tr>
        <tr>
          <td align="right" style="padding:4px 10px 20px 0px;" nowrap>
            Cost:
          </td>
          <td style="padding:4px 0px 0px 0px;"><input type="text" name="part_cost"></td>
        </tr>
        <tr>
          <td align="center" class="buttons" colspan="2">
            <input type="button" value="OK" onclick="apply(this.form)">
            <input type="button" value="Cancel" onclick="cancel()">
          </td>
        </tr>
    </form>
  </body>
</html>
```

You should now be able to click on the "My Report" choice from the Action menu and interactively run the Report.