Intro to QuickApps
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Create a widget</td>
<td>4</td>
</tr>
<tr>
<td>Associate a query with a widget</td>
<td>10</td>
</tr>
<tr>
<td>Interact with other widgets</td>
<td>14</td>
</tr>
<tr>
<td>Organize widgets using a layout</td>
<td>19</td>
</tr>
<tr>
<td>Display data in a graphics widget</td>
<td>22</td>
</tr>
<tr>
<td>Insert a block for the query</td>
<td>27</td>
</tr>
<tr>
<td>Control when widgets are updated</td>
<td>33</td>
</tr>
<tr>
<td>Display a widget conditionally</td>
<td>42</td>
</tr>
<tr>
<td>Do something when a variable changes</td>
<td>49</td>
</tr>
<tr>
<td>Specify conditions for widget validation</td>
<td>55</td>
</tr>
<tr>
<td>Access a particular value in a table</td>
<td>64</td>
</tr>
<tr>
<td>Organize the content</td>
<td>69</td>
</tr>
<tr>
<td>Final QuickApp code</td>
<td>75</td>
</tr>
</tbody>
</table>

© 2017 1010data, Inc. All rights reserved.
1010data's QuickApp framework provides a powerful and versatile set of interactive layouts and widgets for building custom front-end interfaces on top of 1010data's analytics engine.

A QuickApp is essentially a way to specify a simple interactive visual report, dashboard, or application using a set of tags in 1010data's Macro Language. These tags work in conjunction with block code and the other data transformation operations to specify how the QuickApp should accept user inputs and display query data. One of the things that makes QuickApps so powerful is that it's not necessary to write any JavaScript, HTML, CSS, or other client-side code – the whole QuickApp is written in 1010data's Macro Language.

The following Macro Language tags are instrumental in building a QuickApp:

<dynamic>

A <dynamic> element creates an environment of state variables and encapsulates the Macro Language code for both the visual and logical aspects of the QuickApp. This dynamic environment manages the state of the widgets within it and is responsible for updating any widgets that are affected by a change in state. For instance, if a user interaction with one particular widget changes the value of a variable in the environment, the QuickApp will update any other widgets that reference that variable.

<widget>

A <widget> element acts as a container for displaying data and may be used for user input. A <widget> element is used to visually represent a block of Macro Language code. It typically contains a single 1010data query that defines the data that it displays. Widgets can be used as a means for user input, such as an input field or drop-down list, or they can be used to display information to the user, such as in a spreadsheet-like grid or a bar chart.

<layout>

A <layout> element acts as a container for widgets or other layouts and specifies their presentational arrangement. A <layout> element allows you to arrange your widgets. There are a number of predefined layout types that you can use to customize the presentation of your QuickApp. For instance, a tabbed panel allows you to organize your content on separate tabs.

QuickApps can be rendered to a number of output targets, such as a web application, PDF, or Excel workbook. This tutorial will guide you through creating a web-based QuickApp, which will allow the user to specify certain parameters for 1010data queries and display the results of those queries in a web browser.

This tutorial is intended to serve as an introduction to the basic concepts of the QuickApp framework as well as a guide to using the specific mechanisms behind it. It is assumed that you are familiar with the 1010data Macro Language, specifically block code and the data transformation operations. It starts off with a simple example and builds on that with each step. As it progresses, the tutorial addresses more complex topics, which illustrate the substantial power of QuickApps. As you follow along, you will learn how to build a fairly robust and functional QuickApp as well as the logic and structure behind it. It is the intention of this guide to give you the foundation to begin building powerful QuickApps that can satisfy the needs of your analyses on the 1010data platform.
Create a widget

A QuickApp is essentially one or more widgets within a dynamically managed environment. A widget is used to visually represent a block of Macro Language code.

A `<dynamic>` element contains the Macro Language code related to the visual aspects of your QuickApp as well as the query logic that drives both the widget behavior and underlying analyses. The presentational aspect is handled through the use of `<widget>` and `<layout>` elements. The query logic behind the widgets and the analyses are handled via block code and the 1010data data transformation operations, such as `<link>`, `<willbe>`, and `<sel>`.

The simplest QuickApp, therefore, would be a single `<widget>` within a `<dynamic>`.

To create a widget:

1. Open any table (e.g., `pub.demo.weather.stations`).
2. Open the Edit Actions (XML) dialog.
3. Enter the following Macro Language code.

```xml
<dynamic>
  <widget/>
</dynamic>
```
4. Click Apply.

The following single-row table is displayed:

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Data</th>
<th>Tag</th>
<th>Widget</th>
<th>Changed?</th>
<th>Invalid?</th>
<th>Error?</th>
<th>Msg</th>
<th>#Rows</th>
<th>Columns</th>
<th>Rows</th>
<th>Cols</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>VEC: 0</td>
<td>6 bannonce_1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>267</td>
<td>id, name, state, ts.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

This table is the result of running the above 1010data Macro Language code and consists of one row for each widget in the QuickApp. It is used internally to manage the state of the widgets within the `<dynamic>`. The columns in the table contain status information related to each widget. Since the above example only has one widget, the resultant table only has one row.

In order to see the QuickApp in its intended form, it must be rendered to some output target, which may be for the web, as a PDF, or as an Excel workbook.

5. Click View > Show QuickApp.
This action renders the QuickApp for the web, generating the necessary HTML and JavaScript that a browser uses to display each widget in the QuickApp.

Note that because no attributes were specified in the `<widget>` tag, a grid widget is used by default. The grid widget displays its data in tabular form.

A widget can have a query inside its opening and closing `<widget>` tags, which determines the information the widget displays. Since this widget has no query associated with it, and instead uses a self-closing `<widget>` tag, it displays all of the data from the current table, which in this case is `pub.demo.weather.stations`.

By default, widgets operate on the current table, but you can specify the table a widget operates on using the `base_` attribute.

6. Add the `base_` attribute to the widget to display the data from the Product Master table (`pub.demo.retail.prod`).

Note: System attributes are denoted by a trailing underscore. This differentiates them from user-defined attributes, which cannot end in an underscore.

```
<dynamic>
  <widget base_="pub.demo.retail.prod"/>
</dynamic>
```

7. Click Apply.
Now the grid widget displays the contents of pub.demo.retail.prod even though the current table is still pub.demo.weather.stations.

The class_ attribute allows you to specify the form in which the widget data or query results are displayed. As mentioned earlier, since no attributes were specified in the <widget> tag, a grid widget was used by default. If you wanted, you could explicitly specify this using the class_ attribute.

8. Add a class_ attribute to explicitly specify that the data is displayed in a grid widget.

```xml
<widget class_="grid" base_="pub.demo.retail.prod"/>
</dynamic>
```

9. Click Apply.

You can see that explicitly specifying class_="grid" produces the same results as omitting the class_ attribute.

The class_ attribute allows you to display the data in a number of different forms. For example, you might want to display the information in this table as a drop-down list.

10. Change the value of the class_ attribute to display the data from the table in a drop-down list.

```xml
<widget class_="dropdown" base_="pub.demo.retail.prod"/>
```
11. Click **Apply**.

The QuickApp displays the contents of the table as a drop-down menu.

If the table has more than one column, the column second from the left supplies the labels for the items in the drop-down menu. You can see that the items in this drop-down menu correspond to the **Item Description** column, which is the second column of the table specified by the `base_` attribute, `pub.demo.retail.prod`.

You can control the visual aspects or behavior of the widget using attributes specific to the widget class as well as attributes that apply to all widget classes. For instance, you can specify the width of the input field and associated drop-down menu for this widget using the `inputwidth_` attribute.

12. Set the input width of the drop-down widget to 250.

```xml
<widget class_="dropdown" base_="pub.demo.retail.prod"
    inputwidth_="250"/>
```

13. Click **Apply**.

The input width of the drop-down widget is increased to 250 pixels.

---

**Note:** The unit of measurement depends on the target to which the QuickApp is rendered. Since this QuickApp is rendered for the web, the unit of measurement is pixels.

When you select an item from the drop-down menu, the value associated with that selection can be saved in a dynamic variable. The value of a dynamic variable can be shared with other widgets or queries in the QuickApp and can affect their behavior or results. Dynamic variables are declared and given initial values in the opening `<dynamic>` tag.

As previously mentioned, in a drop-down menu, the second column of the table associated with it provides the labels that are displayed in the drop-down list. The first column provides the values. (If the table or
query associated with the drop-down widget has only one column, that column provides both the labels
and the values for the drop-down menu.)

In this example, the drop-down widget displays all the items in the Item Description column. When a
user selects an item from the drop-down menu, the value associated with that selection comes from the
Corresponding row in the SKU column, the first column in pub.demo.retail.prod.

The value attribute is used to specify the dynamic variable where that value can be saved.

14. Specify a dynamic variable for the value attribute of the drop-down widget.

```
<dynamic selection="">
  <widget class_="dropdown" base_="pub.demo.retail.prod"
          inputwidth_="250" value_="@selection"/>
</dynamic>
```

A new dynamic variable, selection, has been added to the <dynamic> and has been set to the
empty string. A value attribute has been added to the drop-down widget and indicates that the value
associated with a selection in the drop-down widget should be assigned to this new dynamic variable.

**Note:** When referring to the dynamic variable in the value attribute (or anywhere else within
the QuickApp), the name of the dynamic variable is preceded by an @ symbol; however, the @
symbol is not used when the variable is declared in the <dynamic>.

To verify that the value of the selected item in the drop-down menu has been assigned to the dynamic
variable, you can add a text widget that displays the value of the dynamic variable. This is a particularly
useful method of debugging as you design your QuickApp.

15. Add a text widget to display the value of the new dynamic variable.

```
<dynamic selection="">
  <widget class_="dropdown" base_="pub.demo.retail.prod"
          inputwidth_="250" value_="@selection"/>
  <widget class_="text" text_="Current selection: {@selection}"/>
</dynamic>
```

**Note:** When referring to the value of a dynamic variable, you must surround the dynamic
variable with curly braces. In this example, in the text attribute of the text widget,
 {@selection} indicates that the value of the dynamic variable selection should be
displayed.

16. Click Apply.

You can see the new text widget to the right of the drop-down widget. (By default, widgets are arranged
horizontally.) Since the value of the dynamic variable selection is set to the empty string, nothing
appears after the Current selection: text.

```
Current selection: 
```

17. Select an item from the drop-down menu (e.g., PEPSI 2 LITER).

The selection appears in the drop-down input field, and its associated value appears in the text widget.

```
PEPSI 2 LITER  Current selection: 969
```

In this example, the item description PEPSI 2 LITER has been selected and appears in the drop-
down input field. The value 969, which comes from the SKU column associated with the selected item
description, has been assigned to the dynamic variable selection and is displayed by the text widget.

Widgets not only allow the user to view the results of a query, but they also allow the user to make
selections and provide input to the QuickApp. The grid, drop-down, and text widgets are just a few of the
many classes of widgets that the QuickApp framework provides.
As you'll see in the following topics, through the use of dynamic variables, the selections you make in one widget can affect the behavior of other widgets in the QuickApp, allowing for a truly interactive experience.

Combining the relative ease of building QuickApps with the power and speed of the 1010data platform creates an unparalleled combination that will take the insights you gain from the analysis of your data to a whole new level.

**Cumulative QuickApp code**

The Macro Language code for the QuickApp up to this point is:

```xml
<dynamic selection=""/>
    <widget class="dropdown" base="pub.demo.retail.prod"
        inputwidth="250" value="@selection"/>
    <widget class="text" text="Current selection: {@selection}"/>
</dynamic>
```
Associate a query with a widget

A query may be used to specify the data that a widget displays. One way to associate a query with a particular widget is to insert the Macro Language code for that query between the opening and closing tags of the `<widget>`.

In *Create a widget* on page 4, the drop-down widget displayed a list of all the item descriptions from the *Product Master* table. When an item was selected from the drop-down menu, the value of the SKU corresponding to the selected item description was assigned to the dynamic variable associated with the `value_` attribute. However, what if you wanted to show a list of all the *department descriptions* from the *Product Master* table in the drop-down list instead and wanted the widget to assign the corresponding *department number* to the dynamic variable? You could add a query to the widget that changes the order of the columns in the table to do just that.

You'll recall that the drop-down widget uses the first two columns from the table associated with the widget. It uses the first column for the values of the drop-down menu and the second column for the labels. To show a list of the department descriptions, whose values are the corresponding department numbers, you could simply add a query that performs a `<color>` to arrange the columns in the desired order. For this example, the `<color>` would arrange the columns such that the column containing the department numbers (`dept`) is the first column and the column containing the department descriptions (`deptdesc`) is the second.

To associate a query with a widget:

1. Add the following query to the grid widget.

   ```<dynamic selection="">
   <widget class_="dropdown" base_="pub.demo.retail.prod"
           inputwidth_="250" value_="@selection">
       <color cols="dept,deptdesc"/>
   </widget>
   <widget class_="text" text_="Current selection: {@selection}"/>
   </dynamic>```

   **Note:** When you specify a query for a particular widget, you may insert the Macro Language code for that query between the start tag and end tag for the `<widget>`. Since the drop-down widget in this example did not have a query associated with it prior to this step, you must change the self-closing `<widget>` tag to a start tag and add an end tag.

2. Click *Apply*.

   ![Dropdown menu with list of department descriptions](image)

   The drop-down menu now displays the value of every item in the `deptdesc` column in the table. You'll notice that there are multiple instances of certain items. You might want to add some logic to the query to include only unique values. In addition, you might want to sort the items in alphabetical order.

3. Modify the query to return only the unique department descriptions in the table along with their associated department numbers, and sort the department descriptions in alphabetical order.
4. Click **Apply**.

The drop-down menu now includes only the unique department descriptions in the table, sorted in alphabetical order.

5. Select an item from the drop-down menu (e.g., **OFFICE SUPPLIES**).

The selection appears in the drop-down input field, and the value of the corresponding department appears in the text widget.

In this example, the department description **OFFICE SUPPLIES** has been selected and appears in the drop-down input field. The value **20**, which comes from the `dept` column associated with the selected department description, has been assigned to the dynamic variable `selection` and is displayed by the text widget.

As you can see, the table that has been specified as the base table for this widget, `pub.demo.retail.prod`, has only five unique departments. To make this tutorial a little more interesting, you can change the base table to a table that has more departments. For the rest of this tutorial, you will use `pub.doc.retail.product`.

To make it easier to reference this table by other widgets or to change this table path in the future, you can create a dynamic variable to hold the table path and use the value of that dynamic variable whenever you need to reference the **Product Master** table.

6. Add a new dynamic variable, **product_master**, and set it to **pub.doc.retail.product**. Set the base attribute for the drop-down widget to the value of this new dynamic variable.
A new dynamic variable has been added to the opening `<dynamic>` tag, and it has been set to `pub.doc.retail.product`. The `base_` attribute for the drop-down widget has been changed to reference the value of that new variable.

**Note:** Remember that dynamic variables are preceded by an `@` symbol when referenced within a QuickApp. To reference the `value` of the dynamic variable, you must surround the name of the dynamic variable (preceded by the `@` symbol) with curly braces. When the QuickApp runs, the values of the dynamic variables will be substituted for the references. In this example, `base_="{@product_master}"` is equivalent to `base_="pub.doc.retail.product"`.

7. Click **Apply**.

The drop-down menu now displays the unique department descriptions in the `pub.doc.retail.product` table, sorted in alphabetical order.

You may want to give the drop-down widget a descriptive label so that the user knows what the drop-down menu contains.

8. Add a `label_` attribute to the drop-down widget.

   ```xml
   <widget class="dropdown" base_="{@product_master}"
       inputwidth_="250" value_="@selection"
       label_="Department:" labelwidth_="75">
   <tabu label="Tabulation on Product Master" breaks="deptdesc">
       <break col="deptdesc" sort="up"/>
       <tcol source="dept" name="dept" fun="first"
           label="First Department"/>
   </tabu>
   </widget>
   ```

9. Click **Apply**.
Intro to QuickApps | Associate a query with a widget | 13

Cumulative QuickApp code

The Macro Language code for the QuickApp up to this point is:

```
<dynamic selection="" product_master="pub.doc.retail.product">
  <widget class="dropdown" base="{@product_master}"
    inputwidth="250" value="@selection"
    label="Department:" labelwidth="75">
    <tabu label="Tabulation on Product Master" breaks="deptdesc">
      <break col="deptdesc" sort="up"/>
      <tcol source="depts" name="depts" fun="first"
        label="First Department"/>
    </tabu>
    <colord cols="dept,deptdesc"/>
  </widget>
  <widget class="text" text="Current selection: {@selection}"/>
</dynamic>
```

The label appears to the left of the drop-down menu.
A powerful aspect of the QuickApp framework is that interactions with one widget can affect the state of other widgets as well as the data that those widgets display. The dynamic environment established by the QuickApp manages the state of all of the widgets and is responsible for updating any widgets that reference a dynamic variable which has changed.

In *Associate a query with a widget* on page 10, you saw that selecting a department description from the drop-down widget automatically changed the department number that was displayed by the text widget. You could take this one step further, and instead of just displaying the department number in a text widget, you could use that value in a 1010data query associated with a grid widget that shows summarized sales data for items in the selected department.

To interact with other widgets:

1. Add a grid widget that uses the dynamic variable modified by the drop-down widget.

```xml
<dynamic_selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail">
  <widget class="dropdown" base_="[@product_master]"
         inputwidth_="250" value_="@selection"
         label_="Department:" labelwidth_="75">
    <tabu label="Tabulation on Product Master" breaks="deptdesc">
      <break col="deptdesc" sort="up"/>
      <tcol source="dept" name="dept" fun="first"
            label="First Department"/>
    </tabu>
  </widget>
  <widget class="text" text_="Current selection: {.@selection}"/>
  <widget class="grid" base_="[@sales_detail]">
    <link table2="[@product_master]" col="sku" col2="sku"
          suffix="_prod" type="select">
      <sel value="dept={.@selection}"/>
    </link>
    <tabu label="Tabulation on Sales Detail" breaks="groupdesc_prod">
      <tcol source="xsales" fun="sum" name="tot_sales"
            label="Sum of Extended Sales" format="type:currency"/>
    </tabu>
    <sort col="tot_sales" dir="down"/>
    <sel value="(groupdesc_prod <> '')"/>
  </widget>
</dynamic>
```

A grid widget has been added that operates on the table `pub.doc.retail.salesdetail`, which is specified via the `base_` attribute using the dynamic variable `sales_detail`. This variable is declared in the opening `<dynamic>` tag.

The query associated with this grid widget first performs a `<link>` operation and links to the table specified by the dynamic variable `product_master`. Inside the `<link>` operation, a `<sel>` operation selects those rows in the *Product Master* table where the department is equal to the value of the dynamic variable `selection` (using the `@{.selection}` notation).

**Note:** In the opening `<dynamic>` tag, the dynamic variable `selection` is set to a valid department number (19). If this dynamic variable is set to the empty string, the QuickApp will produce an error when it runs because, after the dynamic variables have been substituted, the code `<sel value="dept={.selection}"/>` will expand to `<sel value="dept="/>`, which is not valid Macro Language code. By initializing the variable to 19, the code will expand to `<sel value="dept=19"/>`, which is valid.
After the link, the query performs a tabulation to calculate the sum of sales for each group in the `groupdesc` column. It then sorts the totals in descending order and filters out any rows in the `groupdesc` column that do not have a value (or, more specifically, selects those rows in the `groupdesc` column that are not equal to the empty string).

2. **Click Apply.**

<table>
<thead>
<tr>
<th>Current selection: 19</th>
<th>Group Desc</th>
<th>Sum of Extended Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEVERAGE</td>
<td></td>
<td>$980,450.13</td>
</tr>
<tr>
<td>SOFT DRINKS</td>
<td>$593,169.19</td>
<td></td>
</tr>
<tr>
<td>SOFT DRINKS SINGLES</td>
<td>$136,806.40</td>
<td></td>
</tr>
<tr>
<td>ENERGY DRINKS</td>
<td>$88,986.09</td>
<td></td>
</tr>
<tr>
<td>NEW AGE BEVERAGE</td>
<td>$74,771.29</td>
<td></td>
</tr>
<tr>
<td>SPARKLING/ENHANCED WATER</td>
<td>$64,728.28</td>
<td></td>
</tr>
<tr>
<td>SODA MIXERS</td>
<td>$21,466.67</td>
<td></td>
</tr>
<tr>
<td>UNSCANNED BEVERAGE</td>
<td>$342.11</td>
<td></td>
</tr>
</tbody>
</table>

When the QuickApp first runs, the text widget displays **19**, which is the initial value of the dynamic variable `selection`. The input field of the drop-down widget displays **BEVERAGE**, which is the corresponding department description. In addition, the grid widget shows the sum of sales for each of the group descriptions in that department.

When you make a different selection in the drop-down widget, the value of the dynamic variable `selection` changes. When that variable changes, any widgets that reference that variable are automatically updated. In this example, both the text widget and the grid widgets are updated, since they both reference `selection`.

3. **Select a different value in the drop-down widget (e.g., DAIRY DELI).**

<table>
<thead>
<tr>
<th>Current selection: 20</th>
<th>Group Desc</th>
<th>Sum of Extended Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAIRY DELI</td>
<td></td>
<td>$1,678,241.62</td>
</tr>
<tr>
<td>DELI-CHEESE</td>
<td>$373,978.05</td>
<td></td>
</tr>
<tr>
<td>DELI-EGGS</td>
<td>$293,457.34</td>
<td></td>
</tr>
<tr>
<td>DELI-JUICE</td>
<td>$270,017.68</td>
<td></td>
</tr>
<tr>
<td>DELI-YOGURT</td>
<td>$229,770.59</td>
<td></td>
</tr>
<tr>
<td>DELI--BUTTER/HAMSA</td>
<td>$140,223.03</td>
<td></td>
</tr>
<tr>
<td>DELI-SNKS/PROD/BI</td>
<td>$66,035.90</td>
<td></td>
</tr>
<tr>
<td>COTTAGE CHEESE</td>
<td>$65,055.04</td>
<td></td>
</tr>
<tr>
<td>SOUR CREAM</td>
<td>$63,493.15</td>
<td></td>
</tr>
<tr>
<td>DELI-DOUGH PRODUCTS</td>
<td>$60,957.32</td>
<td></td>
</tr>
<tr>
<td>DELI CREAM CHEESE</td>
<td>$55,965.51</td>
<td></td>
</tr>
<tr>
<td>DELI-PUDGINGS/SHSS</td>
<td>$35,748.07</td>
<td></td>
</tr>
<tr>
<td>DELI-CRATED CHEESE</td>
<td>$15,880.88</td>
<td></td>
</tr>
</tbody>
</table>

When the new value is selected in the drop-down widget, the value of `selection` is changed to the corresponding department number. When that variable changes, the text widget is updated to show the new value, and the grid widget is updated to display the results of the query run with the new department. In this example, the text widget displays the department number **20**, which corresponds to the **DAIRY DELI** department description, and the grid widget shows the sum of sales for each of the group descriptions in that department.

Text widgets, like the one in this example, can be useful for debugging purposes as they provide a way to see the values of dynamic variables as you develop your QuickApp. However, you may not want to include them in the final version of your QuickApp. Obviously, you could delete them to remove them from
your QuickApp, but a useful alternative is to use the `<ignore>` element to omit them from the rendered QuickApp. Then, if you need them in the future, all you have to do is remove the `<ignore>` tags.

4. Add an opening and closing `<ignore>` tag around the text widget.

```xml
<dynamic_selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail">
  <widget class_="dropdown" base_="{@product_master}" inputwidth_="250" value_="@selection" label_="Department:" labelwidth_="75">
    <tabu label="Tabulation on Product Master" breaks="deptdesc">
      <tcol source="dept" name="dept" fun="first"
        label="First`Department"/>
    </tabu>
  </widget>
</ignore>

5. Click Apply.

The text widget is not rendered in the QuickApp.

Now that the user can select which department to use for the query, you might want to give them the choice of which column to group by in the tabulation. Instead of a list of all possible columns, you might want to restrict the choices to either the column containing the group description or the column containing the brand. You can add another drop-down widget that uses a `<table>` element to provide an inline table to populate the drop-down menu with these specific items.

6. Add a drop-down widget that allows the user to select the column containing the group description (groupdesc_prod) or the brand (brand_prod).

```xml
<dynamic_selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
  <widget class_="dropdown" base_="{@product_master}" inputwidth_="250" value_="@selection" label_="Department:" labelwidth_="75">
    <tabu label="Tabulation on Sales Detail" breaks="groupdesc_prod">
      <tcol source="xsales" fun="sum" name="tot_sales"
        label="Sum of`Extended Sales" format="type:currency"/>
    </tabu>
  </widget>
</ignore>
A drop-down widget has been added to the QuickApp right after the drop-down used to select the department. The new drop-down widget uses the dynamic variable `aggregate_by` to store the selection (as specified by the `value_` attribute). The dynamic variable `aggregate_by` is declared in the `<dynamic>` and is set to the value `groupdesc_prod`. In addition, the references to `groupdesc_prod` in the query associated with the grid widget have been changed to refer to the value of the `aggregate_by` dynamic variable (using the `{@aggregate_by}` syntax).

The query associated with the new drop-down widget consists of a single `<table>` operation that creates a two-column table. The first column contains the values that the dynamic variable `aggregate_by` will be set to when a selection is made from the drop-down menu, and the second column contains the corresponding labels that will be displayed in the drop-down menu.

7. Click Apply.

8. Select Brand from the Aggregate by drop-down menu.
When the selection is made in the **Aggregate by** drop-down menu, the value of the dynamic variable `aggregate_by` is changed to `brand_prod`. The QuickApp then automatically checks to see which widgets reference that dynamic variable and updates them accordingly. Since the grid widget uses `aggregate_by`, the query associated with it is executed, and the grid widget is updated with the new results.

**Cumulative QuickApp code**

The Macro Language code for the QuickApp up to this point is:

```xml
<dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
  <widget class="dropdown" base="{@product_master}" inputwidth="250" value="@selection" label="Department:" labelwidth="75">
    <tabu label="Tabulation on Product Master" breaks="deptdesc">
      <break col="deptdesc" sort="up"/>
      <tcol source="dept" name="dept" fun="first" label="First Department"/>
    </tabu>
    <color cols="dept,deptdesc"/>
  </widget>
  <widget class="dropdown" value="@aggregate_by" label="Aggregate by:" labelwidth="75" inputwidth="250">
    <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
      <tcol source="xsales" fun="sum" name="tot_sales" label="Sum of Extended Sales" format="type:currency"/>
      <sort col="tot_sales" dir="down"/>
      <sel value="({@aggregate_by} <> '')"/>
    </tabu>
    <widget class="grid" base="{@sales_detail}">
      <link table2="{@product_master}" col="sku" col2="sku" suffix="_prod" type="select">
        <sel value="dept={@selection}"/>
      </link>
      <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
        <tcol source="xsales" fun="sum" name="tot_sales" label="Sum of Extended Sales" format="type:currency"/>
      </tabu>
      <sel value="tot_sales" dir="down"/>
      <sel value="({@aggregate_by} <> '')"/>
    </widget>
  </widget>
</dynamic>
```
Organize widgets using a layout

By default, widgets are arranged horizontally in the order in which they appear within the <dynamic>. However, you can use the <layout> element to change the arrangements of the widgets. The arrangement of widgets alternate vertically and horizontally with each embedded <layout>.

In *Interact with other widgets* on page 14, the two drop-down widgets were arranged horizontally with the grid widget by default. Suppose you wanted the drop-down widgets to be arranged vertically but the grid widget to be arranged horizontally to the right of them. You could put the drop-down widgets inside a <layout> element. The layout and the grid widget would still be arranged horizontally.

To organize widgets using a layout:

1. Place the two drop-down widgets inside the opening and closing tags of a <layout>.

   ```xml
   <dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
     <layout>
       <widget class="dropdown" base="#{product_master}" inputwidth="250" value="#{selection}" label="Department:" labelwidth="75">
         <tabu label="Tabulation on Product Master" breaks="deptdesc">
           <break col="deptdesc" sort="up"/>
           <tcol source="dept" name="dept" fun="first" label="First Department"/>
         </tabu>
         <color cols="dept,deptdesc"/>
       </widget>
       <widget class="dropdown" value="#{aggregate_by}" label="Aggregate by:" labelwidth="75" inputwidth="250">
         <table>groupdesc_prod,Group;brand_prod,Brand</table>
       </widget>
       <ignore>
         <widget class="text" text="Current selection: "#{selection}"/>
       </ignore>
     </layout>
   </dynamic>
   ``

2. Click **Apply**.
The drop-down widgets are now vertically arranged, and the grid widget appears to the right of the layout containing those widgets.

Like widgets, layouts also have attributes that allow you to control what they look like when they are rendered. For instance, you may want to highlight the layout containing the drop-down widgets by giving it a light blue background color and a visible border.

3. Add attributes to the `<layout>` to specify a background color and border.

```html
<dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="{product_master}" inputwidth="250" value="@selection" label="Department:" labelwidth="75">
      ...
    </widget>
  </layout>
</dynamic>
```

4. Click Apply.

The layout containing the drop-down widgets now has a light blue background color and a visible border.

**Cumulative QuickApp code**

The Macro Language code for the QuickApp up to this point is:

```html
<dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="{product_master}" inputwidth="250" value="@selection" label="Department:" labelwidth="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
      </tabu>
    </widget>
  </layout>
</dynamic>
```
<tcol source="dept" name="dept" fun="first" label="First Department"/>
</tabu>
<colord cols="dept,deptdesc"/>

<widget class="dropdown" value="@aggregate_by"
label="Aggregate by:" labelwidth="75" inputwidth="250">
<table>groupdesc_prod,Group;brand_prod,Brand</table>
</widget>
<widget class="text" text="Current selection: {@selection}"/>
</ignore>

<widget class="grid" base="{@sales_detail}">
<link table2="{@product_master}" col="sku" col2="sku"
suffix="_prod" type="select">
<sel value="dept={@selection}"/>
</link>
<tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
<tcol source="xsales" fun="sum" name="tot_sales"
label="Sum of Extended Sales" format="type:currency"/>
</tabu>
<sort col="tot_sales" dir="down"/>
<sel value="({@aggregate_by} <> '')"/>
</widget>
</dynamic>
Display data in a graphics widget

The same data that is displayed in a grid widget can also be displayed graphically. The `class_` attribute determines the form in which the data is displayed.

Up to this point, the grid widget has been used to display the results of its associated query in tabular form. If you wanted to also display the same results as a line chart, all you need to do is make a copy of the grid widget and change the value of the `class_` attribute to `graphics`.

To display data in a graphics widget:

1. Make a copy of the grid widget, paste it after the existing grid widget, then change the value of the `class_` attribute in the copy to `graphics`.

2. Click `Apply`.

The graphics widget is added to the right of the grid widget.

You may want to display the results as a bar chart instead of a line chart, which is the default, and you might also want to change the x-axis to show the name of the group related to the sum of sales for each group.
data point instead of the row number from the tabulation. You can make these customizations to the graphics widget using the `<graphspec>` element.

3. Add a `<graphspec>` element to customize the appearance of the graphics widget.

   **Note:** The `<graphspec>` element is inserted between the opening and closing tags of the `<widget>` and may be placed before or after any query code associated with the widget. In this example, it is inserted after the query.

```xml
<widget class_="graphics" base_="{@sales_detail}">
  <link table2="{@product_master}" col="sku" col2="sku"
       suffix="_prod" type="select">
    <sel value="dept={@selection}"/>
  </link>
  <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}"
        col source="xsales" fun="sum" name="tot_sales"
        label="Sum of `Extended Sales" format="type:currency"/>
  <sort col="tot_sales" dir="down"/>
  <sel value="({@aggregate_by} <> '')"/>
  <graphspec>
    <chart type="bar">
      <data x="{@aggregate_by}" y="tot_sales"/>
      <ticks xrot="45"/>
      <style xaxissize="10" yaxissize="10"/>
    </chart>
  </graphspec>
</widget>
</dynamic>
```

The `<graphspec>` element allows you to customize the graphics widget. It has a child element, `<chart>`, which allows you to specify the type of chart, among other attributes such as the title.

The `<chart>` element in turn has a number of children: `<axes>`, `<data>`, `<legend>`, `<ticks>`, and `<style>`, which allow you to control various aspects of the chart.

The elements used in this example specify that the graph should be displayed as a line chart; the data for the x-axis should come from the column associated with the `aggregate_by` dynamic variable; the data for the y-axis should come from the `tot_sales` column created by the tabulation; the ticks on the x-axis should be rotated 45 degrees; and the font size for the values on both the x-axis and y-axis should be 10px.

Like all the other widgets, you can modify the display-control attributes for the graphics widget. For instance, you might want to make the graphics widget a little wider.

4. Add a `width_` attribute for the graphics widget.

```xml
...<graphspec>
  <chart type="bar">
    <data x="{@aggregate_by}" y="tot_sales"/>
    <ticks xrot="45"/>
    <style xaxissize="10" yaxissize="10"/>
  </chart>
</graphspec>
...```
5. Click **Apply**.

The width of the graphics widget is increased.

6. Select **DAIRY DELI** from the **Department** drop-down menu.

   When a selection is made in the **Department** drop-down, the **selection** dynamic variable is changed. Since both the grid and graphics widgets reference the **selection** dynamic variable, both widgets are automatically updated.

7. Select **Brand** from the **Aggregate by** drop-down menu.

   When a selection is made in the **Aggregate by** drop-down, the **aggregate_by** dynamic variable is changed. Since both the grid and graphics widgets reference the **aggregate_by** dynamic variable, both widgets are automatically updated.
Cumulative QuickApp code

The Macro Language code for the QuickApp up to this point is:

```xml
<dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
  <layout background_="lightblue" border_="10">
    <widget class_="dropdown" base_="{@product_master}" inputwidth_="250" value_="@selection" label_="Department:" labelwidth_="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first" label="First`Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>
    <widget class_="dropdown" value_="@aggregate_by" label_="Aggregate by:" labelwidth_="75" inputwidth_="250">
      <table>groupdesc_prod,Group;brand_prod,Brand</table>
    </widget>
    <ignore>
      <widget class_="text" text_="Current selection: {@selection}"/>
    </ignore>
  </layout>
  <widget class_="grid" base_="{@sales_detail}">
    <link table2="{@product_master} col="sku" col2="sku" suffix="_prod" type="select">
      <sel value="dept={@selection}"/>
    </link>
    <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
      <tcol source="xsales" fun="sum" name="tot_sales" label="Sum of Extended Sales" format="type:currency"/>
    </tabu>
    <sort col="tot_sales" dir="down"/>
    <sel value="(@aggregate_by) <> ''"/>
  </widget>
  <widget class="graphics" base="{@sales_detail}" width="800">
    <link table2="{@product_master} col="sku" col2="sku" suffix="_prod" type="select">
      <sel value="dept={@selection}"/>
    </link>
    <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
      <tcol source="xsales" fun="sum" name="tot_sales" label="Sum of Extended Sales" format="type:currency"/>
    </tabu>
  </widget>
</dynamic>
```
<sort col="tot_sales" dir="down"/>
<sel value="(@aggregate_by) <> ''"/>
<graphspec>
  <chart type="bar">
    <data x="@aggregate_by" y="tot_sales"/>
    <ticks xrot="45"/> 
    <style xaxissize="10" yaxissize="10"/>
  </chart>
</graphspec>
</widget>
</dynamic>
Insert a block for the query

Blocks allow you to share the same query code among multiple widgets without the need to maintain multiple copies of the code.

In Display data in a graphics widget on page 22, a copy of the grid widget was used in its entirety to create the graphics widget. Because of that, both widgets contain the exact same query code. To maintain parity, any change you make to one query would need to be made in the other. Instead of maintaining two copies of the same query, you could use a `<defblock>` element to create a block that contains the Macro Language code for the query and then reference that block in the widgets.

There are two ways you can reference a block in a widget. You could either add an `<insert>` element between the opening and closing `<widget>` tags, or you could reference the block via the `insert_` attribute to `<widget>`. The differences will be discussed in the following steps.

To insert a block for the query:

1. Insert a `<defblock>` element that contains the query code before the opening `<dynamic>` tag.

   ```xml
   <defblock name="sales_by_date">
     <link table2="{@product_master}" col="sku" col2="sku"
           suffix="_prod" type="select">
       <sel value="dept={@selection}"/>
     </link>
     <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
       <tcol source="xsales" fun="sum" name="tot_sales"
             label="Sum of Extended Sales" format="type:currency"/>
     </tabu>
     <sort col="tot_sales" dir="down"/>
     <sel value="({@aggregate_by} <> '')"/>
   </defblock>
   ``

   The `<defblock>` is named `sales_by_date`. This name is used to reference the block.

2. Delete the query code (which appears below in bold) from both the grid and graphics widgets.

   ```xml
   <dynamic selection="19" product_master="pub.doc.retail.product"
            sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
     ...
   </dynamic>
   ``

   Note: Do not delete the `<graphspec>` element from the graphics widget.

   ```xml
   ...
   <widget class="grid" base="{@sales_detail}">
     <link table2="{@product_master}" col="sku" col2="sku"
           suffix="_prod" type="select">
       <sel value="dept={@selection}"/>
     </link>
     <tabu label="Tabulation on Sales Detail" breaks="{@aggregate_by}">
       <tcol source="xsales" fun="sum" name="tot_sales"
             label="Sum of Extended Sales" format="type:currency"/>
     </tabu>
     <sort col="tot_sales" dir="down"/>
     <sel value="({@aggregate_by} <> '')"/>
   </widget>
   ```
3. Replace the deleted query code in both the grid and graphics widgets with an `<insert>` element that references the block.

```
...<widget class="grid" base="{@sales_detail}"
   <insert block="sales_by_date"/>
</widget>
<widget class="graphics" base="{@sales_detail}" width="800">
   <insert block="sales_by_date"/>
</widget>
```

When the QuickApp runs, the Macro Language code from the referenced `sales_by_date` block is essentially substituted in place of each of the `<insert>` elements.

Referencing a block via an `<insert>` element is advantageous when you have other Macro Language code that you want to execute before the block code runs. You could simply add the additional query code before the `<insert>` element within the body of the `<widget>`.

In addition, using the `<insert>` element is useful when you want to insert multiple blocks. You would simply add `<insert>` elements for the blocks in the order you want them to run.

4. Click **Apply**.

Since the code is essentially the same, the results are the same.

Instead of using the `<insert>` element, you could reference the block code using the `insert_` attribute for each `<widget>`.
5. Delete the `<insert>` elements (which appear below in bold) from both the grid and graphics widgets.

```xml
...<widget class="grid" base="{@sales_detail}">
    <insert block="sales_by_date"/>
</widget>
<widget class="graphics" base="{@sales_detail}" width="800">
    <insert block="sales_by_date"/>
    <graphspec>
        <chart type="bar">
            <data x="{@aggregate_by}" y="tot_sales"/>
            <ticks xrot="45"/>
            <style xaxissize="10" yaxissize="10"/>
        </chart>
    </graphspec>
</widget>
</dynamic>
```

6. Add an `insert_` attribute to both the grid and graphics widgets to reference the block code.

```xml
...<widget class="grid" base="{@sales_detail}" insert="sales_by_date"/>
<widget class="graphics" base="{@sales_detail}" width="800" insert="sales_by_date">
    <graphspec>
        <chart type="bar">
            <data x="{@aggregate_by}" y="tot_sales"/>
            <ticks xrot="45"/>
            <style xaxissize="10" yaxissize="10"/>
        </chart>
    </graphspec>
</widget>
</dynamic>
```

When the QuickApp runs, the Macro Language code from the referenced `sales_by_date` block is essentially inserted before any other Macro Language code in the body of the `<widget>`. The block code referenced by the `insert_` attribute will be executed first.

Furthermore, you may only specify one block with the `insert_` attribute. If you want to insert multiple blocks, you should use multiple `<insert>` elements (or specify the one you want to run first with the `insert_` attribute and the others with multiple `<insert>` elements in the body of the `<widget>`).

**Note:** In this example, since you are no longer specifying query code directly within the body of the `<widget>` element for the grid widget, you can use a self-closing `<widget>` tag. However, since the `<graphspec>` element still exists within the body of the `<widget>` element for the graphics widget, you must still use a start tag and end tag for that widget.

7. Click **Apply**.

Since the code is essentially the same, the results are the same.
In this example, the names of the variables in the `<defblock>` (product_master, selection, and aggregate_by) are the same as the dynamic variables in the QuickApp. Because of that, there is no need to pass any variables to the block when referencing it from either the `<insert>` element or the `insert_attribute`. The block code will essentially be inserted where it is referenced, and each variable in the inserted block code will take the value of the dynamic variable with the same name in the QuickApp.

However, there is no requirement that the variables in the `<defblock>` have the same name as the dynamic variables in the QuickApp. If the names are different, you would need to pass in the values as parameters. The next few steps will demonstrate how this is done.

8. Change the names of the variables in the `<defblock>`.

For demonstration purposes, the following changes will be made in the `<defblock>`:

- `product_master` will be renamed to `prod_table`
- `selection` will be renamed to `department`
- `aggregate_by` will be renamed to `group_by`

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="" />
```

Note: You must declare the block variables in the opening `<defblock>` tag. Since you will be passing in the values of these variables when you insert the block, you can set the initial values to the empty string. However, if you wanted to, you could provide initial values. Also, as mentioned earlier, since these are user-defined attributes, they cannot end in an underscore, which differentiates them from the system attributes.

9. Pass the values of the dynamic variables as parameters to the referenced block.
When referencing a block using the `insert_` attribute, you can pass the parameters to the block as attributes in the `<widget>` tag. Note that the three block variables, `prod_table`, `department`, and `group_by` are added to the `<widget>` tag for both the grid and graphics widget, and are set to the values of the respective dynamic variables: `{@product_master}`, `{@selection}`, and `{@aggregate_by}`.

**Note:** If you were using the `<insert>` element to reference the block, you would pass the parameters as attributes in the `<insert>` tag.

10. Click **Apply**.

The results are the same.

---

**Cumulative QuickApp code**

The Macro Language code for the QuickApp up to this point is:

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="">
  <link table2="{@prod_table}" col="sku" col2="sku"
      suffix=" prod" type="select">
    <sel value="dept={@department}"/>
  </link>
  <tabu label="Tabulation on Sales Detail" breaks="{@group_by}"
    tcol source="xsales" fun="sum" name="tot_sales"
  label="Sum of`Extended Sales" format="type:currency"/>
</tabu>
  <sort col="tot_sales" dir="down"/>
  <sel value="{@group_by} <> ' '")"/>
</defblock>
<dynamic selection="19" product_master="pub.doc.retail.product"
sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod">
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="{@product_master}"
      inputwidth="250" value="@selection"
    label="Department:" labelwidth="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first" label="First`Department"/>
      </tabu>
      <color col="dept,deptdesc"/>
    </widget>
    <widget class="dropdown" value="@aggregate_by">
```
label="Aggregate by:" labelwidth="75" inputwidth="250">
    <table>groupdesc Prod, Group; brand Prod, Brand</table>
</widget>
<ignore>
    <widget class="text" text="Current selection: {@selection}"/>
</ignore>
</layout>
<widget class="grid" base="{@sales_detail}"
    insert="sales by date" prod_table="{@product_master}"
    department="{@selection}" group_by="{@aggregate_by}"/>
<widget class="graphics" base="{@sales_detail}" width="800"
    insert="sales by date" prod_table="{@product_master}"
    department="{@selection}" group_by="{@aggregate_by}">
    <graphspec>
        <chart type="bar">
            <data x="{@aggregate_by}" y="tot_sales"/>
            <ticks xrot="45"/>
            <style xaxissize="10" yaxissize="10"/>
        </chart>
    </graphspec>
</widget>
</dynamic>
Control when widgets are updated

By default, a QuickApp automatically updates any and all widgets that reference a dynamic variable whose value has changed. However, this might not be the desired behavior. QuickApps provide a number of mechanisms to give the user more control.

In Display data in a graphics widget on page 22, the grid and graphics widgets were both updated automatically whenever an item was selected in either of the drop-down widgets. The queries in these examples are fairly simple, and the data sets on which they operate are relatively small. However, the cost to run each query for every widget, in terms of both time and system resources, could become quite high if the queries were more complex or the data sets were substantially larger. In this particular example, you might want to allow the user to update the grid and graphics widgets manually once they are finished making all of their selections.

A widget is invalidated when the data associated with it changes. This can happen when there is a change to one or more of the dynamic variables that a widget references. For instance, when the aggregate_by dynamic variable changes, both the grid and graphics widgets are invalidated. The default behavior of a QuickApp is to automatically update any invalidated widgets. However, you can manually control when widgets are updated by setting the mode_attribute to manual in the opening <dynamic> tag or in the <widget> tag for a particular widget. If you set this attribute in the opening <dynamic> tag, all widgets need to be manually updated. If you set this attribute in a <widget> tag, only that particular widget needs to be manually updated. A widget in this state will remain invalidated until it is told to update.

You can control the behavior of an invalidated widget by setting the invmode_attribute for that widget. You can set the invmode_to hide, so that the widget is hidden until it is updated; to block, which prevents any interaction with the widget; or to none, in which case the widget remains in its previous state until it is updated.

You can provide the user with a button to manually update invalidated widgets. When the button is clicked, the invalidated widgets will receive a message to update, and the widgets will run their associated queries and refresh the data that they are displaying.

To control when widgets are updated:

1. Set the mode_attribute in the opening <dynamic> tag to manual.

```
<dynamic selection="19" product_master="pub.doc.retail.product"
  sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
  mode_="manual">
...
```

When the mode_attribute for the <dynamic> is set to manual, any invalidated widgets remain in that state until manually told to update. By default, the widgets are blocked and prevented from any interaction until they receive a message to update. A default message of Please wait... is presented to the user.

2. In the <widget> tag for both the grid and graphics widgets, set the invmsg_attribute to a more descriptive message.

```
<widget class_="grid" base_="{@sales_detail}" insert_="sales_by_date"
  prod_table="{@product_master}" department="{@selection}"
  group_by="{@aggregate_by}"
  invmsg_="Click Run for changes to take effect"/>
<widget class_="graphics" base_="{@sales_detail}" width_="800"
```
3. Click **Apply**.

The QuickApp will not show any visible changes until the widgets are invalidated.

4. **Select Brand** from the **Aggregate by** drop-down menu.

Selecting an item from the drop-down menu invalidates both the grid and graphics widgets, because they both depend on the dynamic variable changed by the drop-down widget. They are both blocked and display the message specified by the `invmsg` attribute.

When widgets are in an invalidated state, they must be told when to update. This can be done by using a button widget with `type_="submit"`. When the button is clicked, it will trigger a refresh of all invalidated widgets.

5. Add a button widget to update invalidated widgets.

**Note:** It makes sense to put the new button in the layout containing the drop-down widgets since it is one of the controls for the state of the QuickApp.

```xml
<dynamic selection="19" product_master="pub.doc.retail.product"
        sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
        mode_="manual">
  <layout background_="lightblue" border_="10">
    <widget class_="dropdown" base_="{@product_master}" inputwidth_="250" value_="@selection"
            label_="Department:" labelwidth_="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first"
              label="First Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>
    <widget class_="dropdown" value_="@aggregate_by"
            label_="Aggregate by:" labelwidth_="75" inputwidth_="250">
      <graphspec>
        <chart type="bar">
          <data x="@aggregate_by" y="tot_sales"/>
          <ticks xrot="45"/>
          <style xaxissize="10" yaxissize="10"/>
        </chart>
      </graphspec>
    </widget>
  </layout>
</dynamic>
```
A <widget> with class_="button" has been added. Because the type_ attribute is set to submit, a message will be sent to all invalidated widgets when this button is clicked.

6. Click Apply.

The Run button is added to the layout under the drop-down widgets.

7. Select DAIRY DELI from the Department drop-down menu.

The grid and graphics widgets are invalidated. They are both blocked and display the specified message.

8. Select Brand from the Aggregate by drop-down menu.

The grid and graphics widgets remain invalidated and blocked.
9. Click Run.

An update message is sent to the invalidated widgets. The widgets run their respective queries with the new values of the dynamic variables in the QuickApp and refresh the data they are displaying.

For certain widgets, you may want their effects to take place immediately. For instance, you may want to have a checkbox in this QuickApp that controls whether or not to display the chart; however, you would not want to force the user to click the Run button upon every interaction with the checkbox.

10. Add a checkbox to control whether or not the chart is visible.

```xml
<dynamic_selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod" mode="manual" display_chart="1">
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="@product_master" inputwidth="250" value="@selection"
      label="Department:" labelwidth="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first" label="First Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>

    <widget class="checkbox" label="Display Chart" value="@display_chart"/>
    <widget class="button" text="Run" type="submit"/>
  </layout>
</dynamic_selection>
```
A new dynamic variable `display_chart` has been added to the `<dynamic>` with an initial value of 1. A checkbox widget has been added to the `<layout>`, which will set the value of `display_chart` depending on whether or not the checkbox is selected. (The default behavior for a checkbox widget is to set the value of the dynamic variable specified by the `value_` attribute to 1 when selected and 0 when the checkbox is cleared.)

A `visible_` attribute, which is set to the value of `display_chart`, has been added to the graphics widget. Therefore, when `display_chart` is set to 1, the graphics widget is visible, and when `display_chart` is set to 0, the graphics widget is hidden.

11. Click Apply.

The **Display Chart** checkbox is added to the layout containing the drop-down widgets.

12. Clear the **Display Chart** checkbox.

Because the graphics widget is dependent on the `display_chart` dynamic variable, and because `mode_` is set to `manual` in the opening `<dynamic>` tag, the graphics widget is invalidated and therefore blocked.
To refresh the widget, you must click the Run button. However, in this instance, the change to the dynamic variable associated with the checkbox does not have any effect on the query associated with the graphics widget. It only dictates whether or not to display the graphics widget. It would make more sense for the graphics widget to update automatically.

One of the ways to control when a widget is invalidated is to set the mode in the opening <dynamic> to auto (or just omit it for the default behavior) and to use the holdfor_ attribute in each <widget> tag. For a particular widget, the holdfor_ attribute allows you to specify a list of dynamic variables that, when changed, will invalidate that widget.

For this example, it would make sense to list the aggregate_by and selection dynamic variables in the holdfor_ attribute for both the grid and graphics widgets (since their queries depend on those two dynamic variables), but not to include the display_chart dynamic variable. That way, when the user selects an item from either the Aggregate by or Department drop-down widgets, the grid and graphics widgets will become invalidated, and the user will need to click Run to refresh them. However, when the checkbox is selected or cleared, the graphics widget will be displayed or hidden automatically.

13. Change the update_ attribute to auto in the opening <dynamic> tag, and set the holdfor_ attribute for the grid and graphics widget to the aggregate_by and selection dynamic variables.

```xml
<dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod" mode_="auto" display_chart="1">

...<widget class_="grid" base_="{@sales_detail}" insert_="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="{@aggregate_by}" invmsg_="Click Run for changes to take effect" holdfor_="@aggregate_by,@selection"/>
<widget class_="graphics" base_="{@sales_detail}" width_="800" insert_="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="{@aggregate_by}" invmsg_="Click Run for changes to take effect" visible_="{@display_chart}" holdfor_="@aggregate_by,@selection">
<branchspec>
<chart type="bar">
<data x="{@aggregate_by}" y="tot_sales"/>
<ticks xrot="45"/>
<style xaxissize="10" yaxissize="10"/>
</chart>
</branchspec>
</widget>
</dynamic>
```
Note: Instead of setting the `mode` attribute to `auto` in the opening `<dynamic>` tag, you could omit the `mode` attribute entirely since the default behavior for the QuickApp is to automatically update all invalidated widgets.

14. Click **Apply**.
15. Clear the **Display Chart** checkbox.
   The chart is hidden.

16. Select the **Display Chart** checkbox.
   The chart is displayed.

17. Select **Brand** from the **Aggregate by** drop-down menu.
   The grid and graphics widget are both invalidated and blocked.

18. Select **DAIRY DELI** from the **Department** drop-down menu.
   The grid and graphics widget remain invalidated and blocked.
19. Click Run.

The grid and graphics widgets are updated to reflect the new values for the `aggregate_by` and `selection` dynamic variables.

**Cumulative QuickApp code**

The Macro Language code for the QuickApp up to this point is:

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="">
  <link table="[@prod_table]" col="sku" col2="sku"
       suffix="" prod type="select">
    <sel value="dept=[@department]"/>
  </link>
  <tabu label="Tabulation on Sales Detail" breaks="[@group_by]">
    <tcol source="xsales" fun="sum" name="tot_sales"
          label="Sum of Extended Sales" format="type:currency"/>
  </tabu>
  <sort col="tot_sales" dir="down"/>
  <sel value="([@group_by] <> '')"/>
</defblock>

<dynamic selection="19" product_master="pub.doc.retail.product"
        sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
        mode="auto" display_chart="1">
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="[@product_master]"
            inputwidth="250" value="@selection"
            label="Department:" labelwidth="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first" label="First Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>
  </layout>
</dynamic>
```
QuickApps allow you to specify a condition that controls whether a particular widget is visible or hidden.

Up until this point, the grid widget has only been used to display the results of its associated query. However, you can add a level of interaction with the grid widget by utilizing the `clickable_` attribute, which specifies the names of one or more columns whose cells will be clickable.

You could imagine that when a brand or group description is clicked in the grid widget, the QuickApp could display a list of all the SKUs related to that item in a list widget. However, you would not want to display the list widget if the user had not clicked anything in the grid widget. You could use the `require_` attribute to the `<widget>` tag to specify that condition. You could then use the `invmode_` attribute to specify the behavior when the condition is not met (i.e., when the widget is invalidated).

To display a widget conditionally:

1. **Add a `clickable_` attribute to the grid widget, which specifies which column is clickable, and assign the value of the clicked cell to a new dynamic variable called `clicked_value`.

   ```
   <dynamic selection="19" product_master="pub.doc.retail.product"
   sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
   mode_="auto" display_chart="1" clicked_value="">
   ...
   </dynamic>
   ...
   <widget class_="grid" base_="{@sales_detail}" insert="sales_by_date"
   prod_table="{@product_master}" department="{@selection}"
   group_by="{@aggregate_by}" invmsg_="Click Run for changes to take effect"
   holdfor_="@aggregate_by,@selection"
   clickable_="{@aggregate_by}" value_="@clicked_value"/>
   ...
   ```

   The new dynamic variable `clicked_value` is declared in the opening `<dynamic>` tag and set to the empty string.

   In addition, the `clickable_` attribute has been added to the grid widget and has been set to the value of the `aggregate_by` dynamic variable, which is initially set to `groupdesc_prod`.

   The `value_` attribute specifies that when an item in the grid widget is clicked, its value will be assigned to the `clicked_value` dynamic variable.

2. **Click Apply**.

   The items in the `groupdesc_prod` column are clickable.
3. Click any item in the clickable column.

   The clicked value is stored in the `clicked_value` dynamic variable.

4. Create a list widget that displays the item descriptions for all the SKUs in the **Product Master** table related to the item clicked in the grid widget.

   ```html
   <widget class="list" base="{@product_master}"
   width="500" maxheight="600">
   <if test="{@aggregate_by = 'brand_prod'}">
      <then>
      <sel value="(brand='{@clicked_value}')"/>
      </then>
   <else>
      <sel value="(groupdesc='{@clicked_value}')"/>
      </else>
   </if>
   <colord cols="sku,description"/>
   <sort col="description" dir="up"/>
   </widget>
   </dynamic>
   ```

   The `base_` attribute, which is set to the value of the `product_master` dynamic variable, specifies that the new list widget uses the **Product Master** table. The query associated with the list widget selects the rows in that table that have either the same brand or the same group description as the item clicked in the grid. It determines this by checking the value of the `aggregate_by` dynamic variable in the `<if>` clause and performs the appropriate `<sel>` operation based on that. It then performs a `<colord>` operation to create a two-column table used by the list widget; the first column contains the SKUs, and
the second column contains the item descriptions. Finally, the <sort> operation ensures that the item
descriptions are displayed in ascending order in the list widget.

The list widget should only be displayed once an item has been clicked in the grid widget. Otherwise, the
list widget should be hidden. You can enforce this behavior by adding a require_attribute to the list
widget which specifies that the clicked_value dynamic variable must be equal to something other than
the empty string. If this condition is not met, the list widget will be invalidated. You can set the invmode_attribute to hide so that the list widget is hidden when it is invalidated.

5. Add a require_attribute to the list widget and set the invmode_attribute to hide.

```xml
<widget class_="list" base_="{@product_master}"
  width_="500" maxheight_="600"
  require_="{@clicked_value <> ''}" invmode_="hide">
  <if test="{@aggregate_by = 'brand_prod'}">
    <then>
      <sel value="(brand='@clicked_value')"/>
    </then>
    <else>
      <sel value="(groupdesc='@clicked_value')"/>
    </else>
  </if>
  <colord cols="sku,description"/>
  <sort col="description" dir="up"/>
</widget>
</dynamic>
```

The require_attribute can take an expression that is either true or false. If the expression is false,
the widget is invalidated. In this example, if the value of the clicked_value dynamic variable is
equal to the empty string, then the expression {clicked_value <> ''} is false, and the widget is
invalidated.

**Note:** The expression specified for the require_attribute must be enclosed within curly
braces.

6. Click Apply.

Since clicked_value is initially set to an empty string in the opening <dynamic> tag, the condition
specified by the require_attribute is not true. Therefore, the list widget is invalidated. Since the
invmode_for the list widget is set to hide, the list widget is hidden.

7. Click an item in the grid widget (e.g., SOFT DRINKS).

The list widget is populated with the item descriptions for those SKUs in the Product Master table that
match the group description clicked in the grid widget.

The list widget appears to the right of the graphics widget. Because the screen is not wide enough to
accommodate it, only half the widget is visible without scrolling to the right.

Since the width of the QuickApp is becoming too wide to fit on the screen without scrolling, it might make
more sense to position the list widget below the grid widget. This can be accomplished by using <layout>
tags.
8. Add `<layout>` tags to organize the widgets.

...
As noted in Organize widgets using a layout on page 19, the arrangement of widgets will alternate vertically and horizontally with each embedded <layout>.

In this example, the outermost set of newly-added <layout> tags creates a container that will be arranged horizontally with the original <layout> containing the drop-down widgets and Run button.

Inside this new <layout>, two more sets of <layout> tags have been added. The containers created by these <layout> tags will be arranged vertically, and their contents will be arranged horizontally.

The container created by the first new innermost set of <layout> tags will appear on top, and inside this container, the grid and graphics widgets will be arranged horizontally. Similarly, the container created by the second new innermost set of <layout> tags will appear below the top container, and the new list widget will appear inside it.

9. Click Apply.

Since the list widget is initially invalidated, it is hidden.

10. Click an item in the grid widget (e.g., SOFT DRINKS).

The list widget appears below the grid widget and displays the item descriptions for those SKUs in the Product Master table that match the group description that was clicked.

Cumulative QuickApp code

The Macro Language code for the QuickApp up to this point is:

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="">
  <link table2="{@prod_table}" col="sku" col2="sku"
    suffix="prod" type="select">
    <sel value="dept={@department}"
    </link>
  <tabu label="Tabulation on Sales Detail" breaks="{@group_by}">
    <tcol source="xsales" fun="sum" name="tot_sales"
      label="Sum of `Extended Sales" format="type:currency"/>
  </tabu>
  <sort col="tot_sales" dir="down"/>
</defblock>
```
<defblock>
<sel value="({@group_by} <> '')"/>
</defblock>

@dynamic selection="19" product_master="pub.doc.retail.product"
sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
mode="auto" display_chart="1" clicked_value=""
<layout background_="lightblue" border_="10">
   <widget class_="dropdown" base_="{@product_master}"
      inputwidth_="250" value_="@selection"
      label_="Department:"
      labelwidth_="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
         <Break col="deptdesc" sort="up"/>
      </tabu>
      <color cols="dept,deptdesc"/>
   </widget>
   <widget class_="dropdown" value_="@aggregate_by"
      label_="Aggregate by:"
      labelwidth_="75"
      inputwidth_="250">
      <table>groupdesc_prod,Group;brand_prod,Brand</table>
   </widget>
   <widget class_="checkbox" label_="Display Chart"
      value_="@display_chart"/>
   <widget class_="button" text_="Run" type_="submit"/>
   <ignore>
      <widget class_="text" text_="Current selection: {@selection}"/>
   </ignore>
</layout>

<layout>
   <widget class_="grid" base_="{@sales_detail}"
      insert_="sales_by_date" prod_table="{@product_master}"
      department="{@selection}" group_by="{@aggregate_by}"
      invmsg_="Click Run for changes to take effect"
      holdfor_="@aggregate_by,@selection"
      clickable_="{@aggregate_by}" value_="@clicked_value"/>
   <widget class_="graphics" base_="{@sales_detail}"
      insert_="sales_by_date" prod_table="{@product_master}"
      department="{@selection}" group_by="{@aggregate_by}"
      invmsg_="Click Run for changes to take effect"
      visible_="{@display_chart}" holdfor_="@aggregate_by,@selection">
      <graphspec>
         <chart type="bar">
            <data x="{@aggregate_by}" y="tot_sales"/>
            <ticks xrot="45"/>
            <style xaxissize="10" yaxissize="10"/>
         </chart>
      </graphspec>
   </widget>
</layout>

<layout>
   <widget class_="list" base_="{@product_master}"
      width_="500" maxheight_="600"
      require_="{@clicked_value <> ''}" invmode_="hide"
      <if test="{@aggregate_by = 'brand_prod'}">
         <sel value="(brand='{@clicked_value}')"/>
      </if>
      <else>
         <sel value="(groupdesc='{@clicked_value}')"/>
      </else>
   </if>
<color cols="sku,description"/>
<sort col="description" dir="up"/>
</widget>
</layout>
</layout>
</dynamic>
Do something when a variable changes

There are times you might want to do something when a particular dynamic variable changes. For instance, you could imagine that if the user interacted with a certain widget in the QuickApp, you would want to set the value of one or more other dynamic variables.

In Display a widget conditionally on page 42, the list widget is only displayed if the user has clicked an item in the grid widget. After that, the widget remains visible. However, if the user changes a value in the Department or Aggregate by drop-down widgets, you might want to hide the list widget again until the user makes a new selection in the grid widget.

This can be accomplished by adding a `<do>` clause that is triggered when either the aggregate_by or selection dynamic variables change. The `<do>` clause will set clicked_value to the empty string. When clicked_value is equal to the empty string, the list widget is invalidated and subsequently hidden.

To do something when a variable changes:

1. Add a `<do>` clause, which is triggered when either the aggregate_by or selection dynamic variables change. This `<do>` clause will set the value of the clicked_value dynamic variable to the empty string.

   ```html
   <dynamic_selection="19" product_master="pub.doc.retail.product"
   sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
   mode="auto" display_chart="1"
   clicked_value="" />
   <do onchange="@aggregate_by,@selection">
     <set clicked_value="" />
   </do>
   <layout background="lightblue" border="10">
     <widget class="dropdown" base="@product_master"
     inputwidth="250" value="@selection"
     label="Department:" labelwidth="75"
   ...```

   The `onchange` attribute in the opening `<do>` tag allows you to specify a comma-separated list of dynamic variables. When the value of any of these variables change, the Macro Language code inside the `<do>` clause is run.

   **Note:** Typically, `<do>` clauses are located at the top of the `<dynamic>` before any `<widget>` elements.

2. Click **Apply**.
3. Click an item in the grid widget (e.g., SOFT DRINKS).
   The list widget appears as expected.
4. Select **Brand** from the **Aggregate by** drop-down menu.

   The grid and graphics widget are both invalidated and blocked, and the list widget is hidden.

5. Click **Run**.

   The grid widget displays a list of all the brands for the specified department.
6. Click an item in the grid widget (e.g., *PEPSI*).

The list widget appears and contains the item descriptions for the SKUs matching the selected brand.

Since the value of the brand or group description that the user clicked in the grid widget is stored in the `clicked_value` dynamic variable, that value can be used to create a label for the list widget, which could be used to identify the contents of the list. The value of this label can be set in a different `<do>` clause, which is triggered when `clicked_value` changes.

7. Create a `<do>` clause, which is triggered when `clicked_value` changes, that sets the label of the list widget using a new dynamic variable named `list_title`.

```html
<do onchange="@aggregate_by,@selection">
  <set clicked_value="" list_title="PRODUCT LIST"/>
</do>
```
A new dynamic variable, `list_title`, has been added to the `<dynamic>` with an initial value of `PRODUCT LIST`.

A `<do>` clause has been added, which is triggered when the value of `clicked_value` changes. This `<do>` clause sets the `list_title` dynamic variable to the value of `clicked_value` followed by the words `PRODUCT LIST`. For instance, if the user clicks `SOFT DRINKS` in the grid widget, the title of the list widget will be `SOFT DRINKS PRODUCT LIST`.

A `label_` attribute, which is set to the value of `list_title`, has been added to the list widget, which will display the label above the widget.

**Note:** If there are multiple `<do>` clauses, they are executed in the order in which they appear in the `<dynamic>`.

8. Click **Apply**.
9. Click an item in the grid widget (e.g., `ENERGY DRINKS`).

The text of the `list_title` dynamic variable now appears above the list widget.
The Macro Language code for the QuickApp up to this point is:

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="">
  <link table2="@prod_table" col="sku" col2="sku">
    suffix="prod" type="select">
      <sel value="dept=[@department]"/>
  </link>
  <tabu label="Tabulation on Sales Detail" breaks="@group_by">
    <tcol source="xsales" fun="sum" name="tot_sales">
      label="Sum of `Extended Sales" format="type:currency"/>
  </tabu>
  <sort col="tot_sales" dir="down"/>
  <sel value="({@group_by} <> '')"/>
</defblock>
<dynamic selection="19" product_master="pub.doc.retail.product"
sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
mode="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST">
  <do onchange="@aggregate_by,@selection">
    <set clicked_value=""/>
  </do>
  <do onchange="@clicked_value">
    <set list_title="{@clicked_value} PRODUCT LIST"/>
  </do>
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="@product_master" inputwidth="250" value="@selection">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first" label="First `Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>
    <widget class="dropdown" value="@aggregate_by" label="Aggregate by:" labelwidth="75" inputwidth="250">
      <table>groupdesc_prod,Group;brand_prod,Brand</table>
    </widget>
  </layout>
</dynamic>
```
<widget class="checkbox" label="Display Chart" value="@display_chart"/>
<widget class="button" text="Run" type="submit"/>
<ignore>
  <widget class="text" text="Current selection: {@selection}"/>
</ignore>
</layout>
<layout>
  <widget class="grid" base="{@sales_detail}"
    insert="sales_by_date" prod_table="{@product_master}"
    department="{@selection}" group_by="{@aggregate_by}"
    invmsg="Click Run for changes to take effect"
    holdfor="{@aggregate_by},{@selection}"
    clickable="{@aggregate_by}" value="@clicked_value"/>
  <widget class="graphics" base="{@sales_detail}" width="800"
    insert="sales_by_date" prod_table="{@product_master}"
    department="{@selection}" group_by="{@aggregate_by}"
    invmsg="Click Run for changes to take effect"
    visible="@display_chart" holdfor="{@aggregate_by},{@selection}"
    <graphspec>
      <chart type="bar">
        <data x="{@aggregate_by}" y="tot_sales"/>
        <ticks xrot="45"/>
        <style xaxissize="10" yaxissize="10"/>
      </chart>
    </graphspec>
  </widget>
</layout>
<layout>
  <widget class="list" base="{@product_master}"
    width="500" maxheight="600"
    require="{@clicked_value <> ''}" invmode="hide"
    label="{@list_title}"
    <if test="{@aggregate_by = 'brand_prod'}">
      <then>
        <sel value="(brand='{@clicked_value}')}}">
      </then>
    </else>
  </widget>
</layout>
Specify conditions for widget validation

While a widget is in an invalidated state, you can hide the widget so that it is not visible to the user. You may want to require that certain conditions are met for a widget to be validated.

For example, when the user clicks an item in the list widget, you might want to display a graphics widget showing a line graph of the sales for that item over a particular time period. However, you would not want to display the graphics widget if the user has not selected anything in the list widget. You can do this by setting the require attribute for the graphics widget to be true only when the dynamic variable associated with the list widget has been set to something.

To specify conditions for widget validation:

1. Add a dynamic variable that holds the value of the item selected in the list widget.

   `<dynamic_selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod" mode="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST" list_selection="">`

   `</dynamic_selection>`

   The list_selection dynamic variable is set to the value of the item selected in the list widget.

   **Note:** This list widget allows the user to select multiple items at the same time, in which case the list_selection dynamic variable is set to a comma-separated list of those items. For the purposes of this tutorial, however, it is assumed that the user selects only one item in the list widget at a time.

2. Add a graphics widget that displays a line chart of the total sales over time for a specified period.

   `<dynamic_selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod" mode="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST" list_selection="" startdate="20140101" enddate="20140131">`

   `</dynamic_selection>`
The `base_` attribute, which is set to the value of the `sales_detail` dynamic variable, specifies that the new graphics widget uses the table that contains the sales data. The `require_` attribute indicates that the widget is invalidated if the `list_selection` dynamic variable is equal to the empty string (i.e., nothing has been selected in the list widget). The `invmode_` attribute specifies that the widget is hidden when invalidated.

The query associated with the new graphics widget first does a selection on a date range specified by two dynamic variables that have been added to the opening `<dynamic>` tag: `startdate` and `enddate`. In this example, these dynamic variables hold static values; however, if you wanted to give the QuickApp more flexibility, you could add a date widget to allow the user to select those values. For this tutorial, the static values are used.

The query then selects the rows in the table where the value in the `sku` column matches the value of the item selected in the list widget and performs a tabulation that calculates the sum of sales for that SKU. The `<willbe>` takes the dates in the `trans_date` column and simply formats them as `MM/DD/YYYY` so that they can be displayed properly on the x-axis of the graph.

Finally, the `<graphspec>` specifies the type of chart (`line`), as well as the label of the x-axis, the data used for the x-axis and y-axis, the degrees of rotation of the ticks on the x-axis, and the font size of the values on the axes.
Because of the condition specified in the `require` attribute for the new graphics widget, the graphics widget will be invalidated and subsequently hidden if the `list_selection` dynamic variable is set to the empty string. Therefore, the new graphics widget will only be displayed once a selection has been made in the list widget.

If the user clicks a different item in the grid widget, the list widget will be refreshed to show the SKUs related to the new item. The line chart related to the previously selected SKU in the list widget should no longer be shown. To hide the graphics widget containing the line chart, you can simply set `list_selection` to the empty string in the `<do>` clause where `onchange_="$@clicked_value"`.

**3.** Set `list_selection` to the empty string in the `<do>` clause where `onchange_="$@clicked_value"`.

```xml
<dynamic selection="19" product_master="pub.doc.retail.product" sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod" mode_="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST" list_selection="" startdate="20140101" enddate="20140131">
  <do onchange_="$@aggregate_by, @selection">
    <set clicked_value=""/>
  </do>
  <do onchange_="$@clicked_value">
    <set list_title="{@clicked_value} PRODUCT LIST"/>
    <set list_selection=""/>
  </do>
</dynamic>
```

Furthermore, just as in *Do something when a variable changes* on page 49, if the user changes a value in the **Department** or **Aggregate by** drop-down widgets, the new graphics widget should be hidden until the user makes a new selection in the list widget. To hide the graphics widget in this case, set `list_selection` to the empty string in the `<do>` clause where `onchange_="$@aggregate_by, @selection"`.

**4.** Set `list_selection` to the empty string in the `<do>` clause where `onchange_="$@aggregate_by, @selection"`.

```xml
<do onchange_="$@aggregate_by, @selection">
  <set clicked_value=""/>
  <set list_selection=""/>
</do>
```

**5.** Click **Apply**.

**6.** Click an item in the grid widget (e.g., **SOFT DRINKS SINGLES**).
The list widget appears as expected.

7. Click an item in the list widget (e.g., *A&W ROOTBEER SINGLE+CRV*).

The graphics widget containing the line chart appears to the right of the list widget.

Changing the selection in the list widget automatically updates the graphics widget containing the line chart, since its query uses the value of `list_selection`.

8. Click a different item in the list widget (e.g., *CANADA DRY ALE+CRV*).

The graphics widget containing the line chart is updated to reflect the new selection in the list widget.
If the user selects a different group description in the grid widget, the list widget is updated accordingly, and the graphics widget containing the line chart is hidden, since list_selection is set to the empty string in the <do> clause where onchange_="@clicked_value".

9. Click an item in the grid widget (e.g., ENERGY DRINKS).

The list widget is updated to show the item descriptions associated with the group description selected in the grid widget, and the graphics widget containing the line chart is hidden.

10. Click an item in the list widget (e.g., AMP ENERGY ELEVATE+CRV).

The graphics widget containing the line chart appears to the right of the list widget.
If the user changes a value in the **Department** or **Aggregate by** drop-down widgets, both the list widget and the graphics widget containing the line chart are hidden because the `clicked_value` and `list_selection` dynamic variables are each set to the empty string in the `<do>` clause where `onchange_="@aggregate_by,@selection"`.

**11. Select Brand from the Aggregate by drop-down menu.**

The grid widget and graphics widget containing the bar chart are invalidated and blocked, and the list widget and graphics widget containing the line chart are hidden.

**12. Click Run.**

The grid widget and graphics widget containing the bar chart are updated, and the list widget and graphics widget containing the line chart are still invalidated and remain hidden.
Cumulative QuickApp code

The Macro Language code for the QuickApp up to this point is:

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="">
  <link table2="{prod_table}" col="sku" col2="sku"
    suffix="_prod" type="select">
    <sel value="dept={department}"/>
  </link>
  <tabu label="Tabulation on Sales Detail" breaks="{@group_by}">
    <tcol source="xsales" fun="sum" name="tot_sales"
      label="Sum of Extended Sales" format="type:currency"/>
  </tabu>
  <sort col="tot_sales" dir="down"/>
  <sel value="{group_by} <> ''"/>
</defblock>
<dynamic selection="19" product_master="pub.doc.retail.product"
  sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
  mode="auto" display_chart="1" clicked_value="" list title="PRODUCT LIST"
  list_selection="" startdate="20140101" enddate="20140131">
  <do onchange="@aggregate_by,@selection">
    <set clicked_value=""/>
    <set list_selection=""/>
  </do>
  <do onchange="@clicked_value">
    <set list_title="{@clicked_value} PRODUCT LIST"/>
    <set list_selection=""/>
  </do>
  <layout background="lightblue" border="10">
    <widget class="dropdown" base="{product_master}"
      inputwidth="250" value="@selection"
      label="Department:" labelwidth="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <tcol source="dept" name="dept" fun="first" label="First Department"/>
      </tabu>
    </widget>
  </layout>
```
<colord cols="dept,deptdesc"/>
</widget>

<widget class="dropdown" value="@aggregate_by"
label="Aggregate by:
labelwidth_="75" inputwidth_="250">
<table>groupdesc_prod,Group;brand_prod,Brand</table>
</widget>

<widget class="checkbox" value="@display_chart">
<ignore>
<widget class="text" text="Current selection: {@selection}"/>
</ignore>
</widget>

<widget class="button" text="Run" type="submit"/>

<ignore>
<widget class="text" text="Current selection: {@selection}"/>
</ignore>

<layout>
<layout>
<widget class="grid" base="{@sales_detail}"
insert_="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="@aggregate_by"
invmsg_="Click Run for changes to take effect"
holdfor_="@aggregate_by,@selection"
clickable_="@aggregate_by" value="@clicked_value"/>
<widget class="graphics" base="{@sales_detail}" width="800"
insert_="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="@aggregate_by"
invmsg_="Click Run for changes to take effect"
visible_="@display_chart" holdfor_="@aggregate_by,@selection">
<graphspec>
<chart type="bar">
<data x="@aggregate_by" y="tot_sales"/>
<ticks xrot="45"/>
<style xaxissize="10" yaxissize="10"/>
</chart>
</graphspec>
</widget>
</layout>

<widget class="list" base="{@product_master}" width="500" maxheight="600"
require_="" invmode="hide"
label_="" value="@list_selection">
<if test="@aggregate_by = 'brand_prod'">
<then>
<sel value="(brand='{@clicked_value}')"/>
</then>
<else>
<sel value="(groupdesc='{@clicked_value}')"/>
</else>
</if>
<colord cols="sku,description"/>
<sort col="description" dir="up"/>
</widget>

<widget class="list" base="{@product_master}"
width="800" height="400" invmode="hide"
label_="noitem" value="@list_selection">
<if test="(@list_selection <> '')">
<sel value="between(trans_date;{@startdate};{@enddate})"/>
<sel value="(sku={@list_selection})"/>
<tabu label="Tabulation on Sales Detail" breaks="trans_date,sku">
<tcol source="xsales" fun="sum" name="tot_sales_by_date" label="Sum of Extended Sales" format="type:currency"/>
<willbe name="date" value="trans_date" format="type:date4y"/>
<graphspec>
  <chart type="line">
  <axes xlabel="Date"/>
  <data x="date" y="tot_sales_by_date"/>
  <ticks xrot="45"/>
  <style xaxissize="10" yaxissize="10"/>
  </chart>
</graphspec>
</if>
</widget>
</layout>
</layout>
</dynamic>
There are times when you might want to get the value of a specific cell in a table and save it in a dynamic variable.

For instance, you might want to add a title to the line chart that was created in [Specify conditions for widget validation](#) on page 55 which shows the item description that was selected in the list widget. However, since the value of the `list_selection` dynamic variable is the SKU and not the item description, you need to look up that SKU in the Product Master table to get the associated item description. Once you find the value in the table, you can save it to a dynamic variable in a `<do>` clause. The specific cell can be specified using the `row_` and `col_` attributes in the `<do>` tag, and the dynamic variable can be specified using the `value_` attribute.

To access a particular value in a table:

1. Add a `<do>` clause that sets a dynamic variable to the value of a specific cell, and use that dynamic variable as the title for the line graph.

```xml
<do onchange_="@aggregate_by,@selection">
    <set clicked_value=""/>
    <set list_selection=""/>
</do>
<do onchange_="@clicked_value">
    <set list_title="{@clicked_value} PRODUCT LIST"/>
    <set list_selection=""/>
</do>
<do onchange_="@list_selection" base_="{@product_master}"
    value_="@item_title" row_="1" col_="1"
    when_="{list_selection <> ''}">
    <sel value="sku={@list_selection}"/>
    <colord cols="description"/>
</do>
```

```xml
<layout background_="lightblue" border_="10">
    <widget class="dropdown" base_="{@product_master}"
        inputwidth_="250" value_="@selection"
        label_="Department:" labelwidth_="75">
    ...
</widget>
```

```xml
<widget class="graphics" base_="{@sales_detail}"
    width_="800" height_="400" invmode_="hide"
    require_="{list_selection <> ''}">
    <if test="{list_selection <> ''}"
        <sel value="between(trans_date;{@startdate};{@enddate})"/>
        <sel value="(sku={@list_selection})"/>
        <tabu label="Tabulation on Sales Detail" breaks="trans_date,sku">
            <tcol source="xsales" fun="sum" name="tot_sales_by_date"
                label="Sum of `Extended`Sales" format="type:currency"/>
        </tabu>
        <willbe name="date" value="trans_date" format="type:date4y"/>
    </if>
    <graphspec>
        <chart type="line" title="@item_title">
            <axes xlabel="Date"/>
            <data x="date" y="tot_sales_by_date"/>
```
A new dynamic variable, item_title, has been added to the opening tag of the <dynamic> and set to SALES OVER TIME. (This value will be changed by the <do> clause you’re adding, so it doesn’t matter particularly what this initial value is.)

A <do> clause has been added that runs when the list_selection dynamic variable changes. Whenever the user selects something in the list widget, the Macro Language code in this <do> clause runs.

The base_attribute in the <do> tag sets the base table to the value of the product_master dynamic variable.

The query in the <do> clause selects the row in the Product Master where the value in the sku column equals the value of the list_selection dynamic variable. The <colord> then ensures that the item description is in the first column. This query results in a table with one row and one column.

The row_ and col_ attributes specify that the value in the first row and first column is stored in the dynamic variable specified by the value_attribute, which in this example is item_title.

The when_attribute specifies that the <do> clause should only run when the value of the list_selection dynamic variable is not equal to the empty string. This is done to prevent an error in the <sel> operation, which selects rows based on the value of list_selection.

Finally, a title has been added to the <chart> element in the <graphspec> for the line chart. The title is set to the value of the item_title dynamic variable.

2. Click Apply.
3. Click an item in the grid widget (e.g., SOFT DRINKS SINGLES).

The list widget is displayed.

4. Click an item in the list widget (e.g., A&W ROOTBEER SNGLE+CRV).

The graphics widget containing the line chart now appears with the title above it.
Cumulative QuickApp code

The Macro Language code for the QuickApp up to this point is:

```html
<defblock name="sales_by_date" prod_table="" department="" group_by=""> 
  <link table2="[@prod_table]" col="sku" col2="sku" 
    suffix="/prod" type="select"> 
    <sel value="dept={@department}"/>
  </link>
  <tabu label="Tabulation on Sales Detail" breaks="[@group_by]"> 
    <tcol source="xsales" fun="sum" name="tot_sales" 
      label="Sum of Outer Sales" format="type:currency"/>
  </tabu>
  <sort col="tot_sales" dir="down"/>
  <sel value="({@group_by} <> '')"/>
</defblock>

<dynamic selection="19" product_master="pub.doc.retail.product" 
  sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod" 
  mode="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST" 
  list_selection="" startdate="20140101" enddate="20140131" 
  item_title="SALES OVER TIME"> 
  <do onchange="@aggregate_by,@selection"> 
    <set clicked_value=""/>
    <set list_selection=""/>
  </do>
  <do onchange="@clicked_value"> 
    <set list_title="{@clicked_value} PRODUCT LIST"/>
    <set list_selection=""/>
  </do>
  <do onchange="@list_selection" base="{@product_master}" 
    value="@item_title" row="1" col="1" 
    when="{@list_selection <> '}'"> 
    <sel value="sku={@list_selection}"/>
    <colord cols="description"/>
  </do>
</dynamic>

<layout background="lightblue" border="10"> 
  <widget class="dropdown" base="{@product_master}" 
    inputwidth="250" value="@selection" 
  </widget>
</layout>
```
label = "Department:" labelwidth = "75">
  <tabu label="Tabulation on Product Master" breaks="deptdesc">
    <break col="dept" sort="up"/>
    <tcol source="dept" name="dept" fun="first" label="First Department"/>
  </tabu>
  <color cols="dept,deptdesc"/>
</widget>

<widget class="dropdown" value="@aggregate_by">
  label = "Aggregate by:" labelwidth = "75" inputwidth = "250">
    <table>groudesc_prod,Group;brand_prod,Brand</table>
  </widget>

<widget class="checkbox" label="Display Chart" value="@display_chart"/>
<widget class="button" text="Run" type="submit"/>
<ignore>
  <widget class="text" text="Current selection: {@selection}"/>
</ignore>
</layout>

<layout>
  <layout>
    <widget class="grid" base="{@sales_detail}"
      insert="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="{@aggregate_by}"
      invmsg="Click Run for changes to take effect"
      holdfor="@aggregate_by,@selection"
      clickable="@aggregate_by" value="@clicked_value"/>

  <widget class="graphics" base="{@sales_detail}" width="800"
    insert="sales_by_date" prod_table="{@product_master}"
    department="{@selection}" group_by="{@aggregate_by}"
    invmsg="Click Run for changes to take effect"
    visible="@display_chart"
    holdfor="@aggregate_by,@selection">
    <graphspec>
      <chart type="bar">
        <data x="@aggregate_by" y="tot_sales"/>
        <ticks xrot="45"/>
        <style xaxissize="10" yaxissize="10"/>
      </chart>
    </graphspec>
  </widget>
</layout>

<layout>
  <widget class="list" base="{@product_master}"
    width="500" maxheight="600"
    require="@clicked_value <> ''" invmode="hide"
    label="@list_title" value="@list_selection"/>
  <if test="@aggregate_by = 'brand_prod'">
    <then>
      <sel value="(brand='@clicked_value')"/>
    </then>
    <else>
      <sel value="(groudesc='@clicked_value')"/>
    </else>
  </if>
  <color cols="sku,description"/>
  <sort col="description" dir="up"/>
</widget>

<widget class="graphics" base="{@sales_detail}"
  width="800" height="400" invmode="hide"
  require="@list_selection <> ''"/>
  <if test="@list_selection <> ''">
Organize the content

You can organize the content of your QuickApp using any of the predefined layout types, such as a splitter layout, which allows you to divide the QuickApp into resizable areas.

The QuickApp framework provides a number of predefined layout types. For instance, a tabbed panel allows you to organize your content on separate tabs. A collapsible layout presents information in sections that can be expanded or collapsed by the user.

A splitter layout allows you to divide your content into multiple areas that can be displayed simultaneously. You can specify the percentage of space allocated to each area contained within a splitter layout, and the dividers that separate the sections can be moved by the user to adjust the space dynamically. In addition, each section can be expanded or collapsed completely.

For this example, suppose you want to separate the area that contains the drop-down widgets and Run button from the grid, list, and both graphics widgets. A splitter layout would make it possible for the user to collapse the panel that contains the drop-down widgets once the selections have been made.

To organize the content using a splitter layout:

1. Add an opening `<layout>` tag with `type_="splitter"` before the first `<layout>` element and add a closing `<layout>` tag after the last `<layout>` element.

```xml
<layout type_="splitter" width_="100%" height_="100%" arrange_="h">
  <layout background_="lightblue" border_="10" width_="20%">
    <widget class_="dropdown" base_="@product_master"
      inputwidth_="250" value_="@selection"
      label_="Department:" labelwidth_="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tcol source="dept" name="dept" fun="first"
          label="First Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>
  </layout>
  <widget class_="dropdown" value_="@aggregate_by"
    label_="Aggregate by:" labelwidth_="75" inputwidth_="250">
    <table>groupdesc_prod,Group;brand_prod,Brand</table>
  </widget>
</layout>
```
<widget class="checkbox" label_="Display Chart"
    value_="@display_chart"/>
<widget class="button" text_="Run" type_="submit"/>
<ignore>
    <widget class="text" text_="Current selection: {@selection}"/>
</ignore>
</layout>
(layout width="80%">
<layout>
    <widget class="grid" base_="{@sales_detail}"
        insert_="sales_by_date" prod_table="{@product_master}"
        department="{@selection}" group_by="{@aggregate_by}"
        invmsg_="Click Run for changes to take effect"
        holdfor_="#aggregate_by,#selection"
        clickable_="#aggregate_by" value_="@clicked_value"/>
<widget class="graphics" base_="{@sales_detail}" width="800"
    insert_="sales_by_date" prod_table="{@product_master}"
    department="{@selection}" group_by="{@aggregate_by}"
    invmsg_="Click Run for changes to take effect"
    visible_="#display_chart" holdfor_="#aggregate_by,#selection">
    <graphspec>
        <chart type="bar">
            <data x="#aggregate_by" y="tot_sales"/>
            <ticks xrot="45"/>
            <style xaxissize="10" yaxissize="10"/>
        </chart>
    </graphspec>
</widget>
</layout>
</layout>
<layout>
    <widget class="list" base_="{@product_master}"
        width="500" maxheight="600"
        require_="@clicked_value <> '<'" invmode="hide"
        label_="@list_title" value_="@list_selection">
        <if test="@aggregate_by = 'brand_prod'">
            <then>
                <sel value="(brand='@clicked_value')"/>
            </then>
        </else>
        <sel value="(groupdesc='@clicked_value')"/>
    </if>
    <colord cols="sku,description"/>
    <sort col="description" dir="up"/>
</widget>
</layout>
<widget class="graphics" base_="{@sales_detail}"
    width="800" height="400" invmode="hide"
    require_="@list_selection <> '<'"/>
    <if test="@list_selection <> '<'">
        <sel value="between(trans_date;{@startdate};{@enddate})"/>
        <sel value="(sku='@list_selection')"/>
        <tabu label="Tabulation on Sales Detail"
            breaks="trans_date,sku">
            <tcol source="xsales" fun="sum" name="tot_sales_by_date"
                label="Sum of Extended Sales" format="type:currency"/>
        </tabu>
        <willbe name="date" value="trans_date" format="type:date4y"/>
    </if>
    <axes xlabel="Date"/>
    <data x="date" y="tot_sales_by_date"/>
    <ticks xrot="45"/>
    <style xaxissize="10" yaxissize="10"/>
A new `<layout>` with `type="splitter"` has been added around all the other layouts in the QuickApp. The `width` and `height` attributes are set to 100%, which tells the QuickApp that the splitter layout should take up all available space on the screen. The `arrange` attribute specifies that the items in the splitter layout should be arranged horizontally, overriding whatever the default arrangement might be.

In addition, a `width` attribute has been added to each of the `<layout>` elements contained within the splitter layout. The values specify that the first `<layout>` is allocated 20% of the space inside the splitter layout, and the second gets 80%.

2. Click **Apply**.

The content is displayed within a splitter layout.

You can grab the handle on the splitter ( ) and move the divider to the left or right to resize the panels, or you use the arrows to collapse or expand the sections.

3. Click the **Collapse** icon ( ) above the handle on the splitter.

The left container is fully collapsed.
4. Click the **Expand** icon (↑) on the splitter.

   The left container returns to its prior width.

---

**Cumulative QuickApp code**

The Macro Language code for the QuickApp up to this point is:

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by=""/>
<link table2="{@prod_table}" col="sku" col2="sku"
suffix="_prod" type="select"/>
<sel value="dept={@department}"/>
</link>
<tabu label="Tabulation on Sales Detail" breaks="{@group_by}"
<tc source="xsales" fun="sum" name="tot_sales"
  label="Sum of Extended Sales" format="type:currency"/>
</tabu>
<sort col="tot_sales" dir="down"/>
<sel value=">({@group_by} <> '')"/>
</defblock>
<dynamic selection="19" product_master="pub.doc.retail.product"
sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
mode="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST"
list_selection="" startdate="20140101" enddate="20140131"
item_title="SALES OVER TIME">
<do onchange="@aggregate_by,@selection">
  <set clicked_value=""/>
  <set list_selection=""/>
</do>
<do onchange="@clicked_value">
  <set list_title="{@clicked_value} PRODUCT LIST"/>
  <set list_selection=""/>
</do>
<do onchange="@list_selection" base="{@product_master}"
  value="@item_title" row="1" col="1"
  when="@list_selection <> ''">
  <sel value="sku={@list_selection}"/>
  <colord cols="dept,deptdesc"/>
</do>
<layout type="splitter" width="100%" height="100%" arrange="h">
  <layout background="lightblue" border="10" width="20%">
    <widget class="dropdown" base="{@product_master}"
      inputwidth="250" value="@selection"
      label="Department: " labelwidth="75">
      <tabu label="Tabulation on Product Master" breaks="deptdesc">
        <break col="deptdesc" sort="up"/>
        <tc source="dept" name="dept" fun="first"
          label="First Department"/>
      </tabu>
      <colord cols="dept,deptdesc"/>
    </widget>
    <widget class="dropdown" value="@aggregate_by"
      label="Aggregate by:" labelwidth="75" inputwidth="250">
      <table>groupdesc_prod,Group;brand_prod,Brand</table>
    </widget>
    <widget class="checkbox" label="Display Chart" value="@display_chart"/>
    <widget class="button" text="Run" type="submit"/>
    <ignore>
      <widget class="text" text="Current selection: {@selection}"/>
    </ignore>
  </layout>
  <layout width="80%">
    <layout>
      <widget class="grid" base="{@sales_detail}" insert="sales_by_date" prod_table="" department="" group_by="" invmsg="Click Run for changes to take effect" holdfor="" clickable="" value="@clicked_value"/>
    </layout>
  </layout>
</layout>
insert = "sales_by_date" prod_table="{@product_master}" 
department="{@selection}" group_by="{@aggregate_by}" 
invmsg = "Click Run for changes to take effect" 
visible = "@display_chart" holdfor="@aggregate_by,@selection">
  <graphspec>
    <chart type="bar">
      <data x="{@aggregate_by}" y="tot_sales"/>
      <ticks xrot="45"/>
      <style xaxissize="10" yaxissize="10"/>
    </chart>
  </graphspec>
  <widget class="list" base="{@product_master}"
    width="500" maxheight="600" 
    require="{@clicked_value <> ''}" invmode="hide" 
    label="{@list_title}" value="@list_selection">
    <if test="{@aggregate_by = 'brand_prod'}">
      <then>
        <sel value="(brand='{@clicked_value}')"/>
      </then>
      <else>
        <sel value="(groupdesc='{@clicked_value}')"/>
      </else>
    </if>
  </widget>
  <widget class="list" base="{@sales_detail}"
    width="800" height="400" invmode="hide" 
    require="{@list_selection <> ''}"
    invmode="hide"
    require="@list_selection <> ''">
    <if test="{@list_selection <> ''}"
      <sel value="between(trans_date;{@startdate};{@enddate})"/>
      <sel value="(sku={@list_selection})"/>
    </if>
    <tabu label="Tabulation on Sales Detail" 
      breaks="trans_date,sku">
      <tcol source="xsales" fun="sum" name="tot_sales_by_date" 
        label="Sum of Extended Sales" format="type:currency"/>
    </tabu>
    <willbe name="date" value="trans_date" format="type:date4y"/>
  </widget>
</layout>
Final QuickApp code

A QuickApp consists of a single `<dynamic>` tag that contains one or more `<widget>` and `<layout>` tags and other Macro Language elements.

To run the final version of the QuickApp, paste the following Macro Language code into the **Edit Actions** (XML) dialog, click **Apply**, and then click **View > Show QuickApp**.

```xml
<defblock name="sales_by_date" prod_table="" department="" group_by="">
    <link table2="{@prod_table}" col="sku" col2="sku"
        suffix="_prod" type="select">
        <sel value="dept={@department}"/>
    </link>
    <tabu label="Tabulation on Sales Detail" breaks="{@group_by}"
        label="Sum of Extended Sales" format="type:currency">
        <tcol source="xsales" fun="sum" name="tot_sales"
            label="Sum of Extended Sales"/>
        <sort col="tot_sales" dir="down"/>
        <sel value="{@group_by} <> ''"/>
    </tabu>
</defblock>

<dynamic selection="19" product_master="pub.doc.retail.product"
    sales_detail="pub.doc.retail.salesdetail" aggregate_by="groupdesc_prod"
    mode="auto" display_chart="1" clicked_value="" list_title="PRODUCT LIST"
    list_selection="" startdate="20140101" enddate="20140131"
    item_title="SALES OVER TIME">
    <do onchange="@aggregate_by,@selection">
        <set clicked_value=""/>
        <set list_selection=""/>
    </do>
    <do onchange="@clicked_value">
        <set list_title="{@clicked_value} PRODUCT LIST"/>
        <set list_selection=""/>
    </do>
    <do onchange="@list_selection" base="@item_master"
        value="@item_title" row="1" col="1"
        when="@list_selection <> ''">
        <sel value="sku={@list_selection}"/>
        <colord cols="description"/>
    </do>

    <layout type="splitter" width="100%" height="100%" arrange="h">
        <widget class="dropdown" base="@product_master"
            inputwidth="250" value="@selection"
            label="Department:" labelwidth="75">
            <tabu label="Tabulation on Product Master" breaks="deptdesc">
                <break col="deptdesc" sort="up"/>
                <tcol source="dept" name="dept" fun="first"
                    label="First Department"/>
            </tabu>
            <colord cols="dept,deptdesc"/>
        </widget>
        <widget class="dropdown" value="@aggregate_by"
            label="Aggregate by:" labelwidth="75" inputwidth="250">
            <table>groupdesc_prod,Group;brand_prod,Brand</table>
        </widget>
        <widget class="checkbox" label="Display Chart" value="@display_chart"/>
        <widget class="button" text="Run" type="submit"/>
    </layout>
</dynamic>
```
<widget class="text" text="Current selection: {@selection}"/>

<layout width="80%">

<widget class="grid" base="{@sales_detail}" insert="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="{@aggregate_by}" invmsg="Click Run for changes to take effect" holdfor="@aggregate_by,@selection" clickable="@aggregate_by" value="@clicked_value"/>

<widget class="graphics" base="{@sales_detail}" width="800" insert="sales_by_date" prod_table="{@product_master}" department="{@selection}" group_by="{@aggregate_by}" invmsg="Click Run for changes to take effect" visible="@display_chart" holdfor="@aggregate_by,@selection">

<graphspec>
<chart type="bar">
<data x="@aggregate_by" y="tot_sales"/>
<ticks xrot="45"/>
<style xaxissize="10" yaxissize="10"/>
</chart>
</graphspec>
</widget>
</layout>

<layout>
<widget class="list" base="{@product_master}" width="500" maxheight="600" require="[@clicked_value <> '']" invmode="hide" label="{@list_title}" value="@list_selection">
<if test="@aggregate_by = 'brand_prod'">
<then>
<sel value="(brand='@clicked_value')"/>
</then>
</if>
<else>
<sel value="(groupdesc='@clicked_value')"/>
</else>
</widget>
<color cols="sku,description"/>
<sort col="description" dir="up"/>
</layout>

<widget class="graphics" base="{@sales_detail}" width="800" height="400" invmode="hide" require="[@list_selection <> '']">
<if test="[@list_selection <> '']">
<sel value="between(trans_date;{@startdate};{@enddate})"/>
<sel value="(sku={@list_selection})"/>
<tabu label="Tabulation on Sales Detail" breaks="trans_date,sku">
<tcol source="xsales" fun="sum" name="tot_sales_by_date" label="Sum of Extended Sales" format="type:currency"/>
</tabu>
</if>
<willbe name="date" value="trans_date" format="type:date4y"/>
</graphspec>
</widget>
</layout>
</if>
</widget>
</layout>
</layout>
</layout>
</dynamic>