Working with Tables

- Creating a Page with a Table
- Adding the Sorting Capability to a Screen-Based Table
- Adding the Sorting Capability to a Procedure based Table
- Changing Table Layout for Instant HTML Pages
- Retrieving Values from a Selected Row within a Table
- Customizing the Table's Display

Creating a Page with a Table

Tables are a part of the Screen editor now, and no longer exist as stand alone entity. In order to define a screen based table, open the screen and click the table tab then click the "Create table" link, ApplinX will automatically create a table that contains all multiple fields on the screen.

The framework provides extra functionality in terms of displaying the mapped table entity. The framework will automatically map a declared screen table to an HTML table with the same ID. The ApplinX Framework will duplicate the first data row with real values from the host. Child elements of the row (such as td/input) are bound to the relevant application field according to the ID attributes.

When generating a Web page for a screen containing a table, code representing the table is generated automatically. When creating a page manually, it is necessary to create a table and bind it to the ApplinX Table.

To create a page with a table (JSP):

1. In the Web page corresponding to the screen containing the ApplinX Table, add an HTML table with an ID that is the name of either an ApplinX screen table or the output variable of a procedure (usually a data structure array) representing the Table: `<gx:table id="<ApplinX Table name>">` Where `<ApplinX Table name>` is the name of the ApplinX Table/Variable name.

2. Add a representation for the table headers:

```
<tr>
       <th> Item Number </th>
       <th> Item Description </th>
</tr>
```

3. Add one row representing the table's data. The ID's should correspond to the names of the ApplinX table columns/data structure attributes, however the order of the columns does not have to be the same:

```
<tr>
        <td id="<column1_name>">dummy data</td>
        <td><input id="<column2_name>" size="10" value="sample data"></td>
</tr>
</gx:table>
```

4. Design the headers and sample columns as required. The sample row serves as a template, and is duplicated in runtime for all the rows of the table.
5. It is possible to change specific rows or cells during the row duplication process.
6. For more options, refer to Working with Tables.

To create a page with a table (.NET):

1. In the Web page corresponding to the screen containing the ApplinX Table, add an HTML table with an ID that is the name of either an ApplinX screen table or the output variable of a procedure (usually a data structure array) representing the Table:`<table id="<ApplinX Table name>" runat="server">` . Where `<ApplinX Table name>` is the name of the ApplinX Table.
2. Add a representation for the table headers:

```
<tr>
          <th> Item Number </th>
          <th> Item Description </th>
</tr>
```

3. Add one row representing the table's data. The ID's should correspond to the names of the ApplinX table columns/data structure attributes, however the order of the columns does not have to be the same:

```
<tr>
          <td id="<column1_name>">dummy data</td>
          <td><input id="<column2_name>" size="10" runat="server" value="sample data"></td>
</tr>
</Table>
```

4. Design the headers and sample columns as desired. The sample row serves as a template, and is duplicated in runtime for all the rows of the table.
5. It is possible to change specific rows or cells during the row duplication process.
6. For more options, refer to Working with Tables.

Sample code can be found in the Composite demo: BrowseProposals.jsp/aspx - a page containing an HTML table bound to an ApplinX screen table.

Adding the Sorting Capability to a Screen-Based Table

It is possible to use the ApplinX Table API to add server-side sorting capabilities to ApplinX Tables in generated Web pages.

To add the sorting capability to a table (JSP):

1. Generate a JSP page for the host screen, containing the table (using the option generate code).
2. In the generated JSP page, add two hidden `gx:input` elements. They will be used to pass the values of the sorted column and will determine whether the sorting order should be ascending or descending:

```
<gx:input type="hidden" id="sortCol"/>
    <gx:input type="hidden" id="isAsc"/>
```

3. In the generated JSP page, add the following JavaScript function. It will set the selected values into the hidden input fields and submit the page:

```
<script>
function sort(colName,isAscending){
document.getElementById('sortCol').value = colName;
       document.getElementById('isAsc').value = isAscending;
       gx_postBack('sort');
}
</script>
```

4. In the JSP code of the table, add calls to the JavaScript function in the headers of all columns that should be sorted columns:

```
<th>
         Item Name
<img src="images/up.gif" onclick="sort('ItemName','true');"
style="cursor:hand"/>
<img src="images/down.gif" onclick="sort('ItemName','false');"
style="cursor:hand"/>
</th>
```

   Where '`ItemName`' in the sort function sort('ItemName','true'); is the name of the column in the ApplinX table definition. The images for the arrows (*up.gif* and *down.gif*) can be copied from the *Sabrafood /images* folder.

5. In the generated code class of the page (`web-inf\classes\contexts\screenName.java`) add the sort function. It will retrieve the values from the request, perform the sort function and build the new sorted table:

```
 public void sort(){
GXITable gx_table = null;
try {
gx_table
       = getGXSession().getScreen().getTables()[0];
} catch (GXGeneralException e) {
gx_handleSessionError(e);
return;
}
       boolean isAscending = false;
       if ("true".equals(getRequest().getParameter("isAsc"))){
       isAscending = true;
   }
gx_table.sort(getRequest().getParameter("sortCol"),isAscending);
getTagsAccesor().setTagTable("ItemsListTable",gx_table,getGXAppConfig().getTableBuildConfig(),this);
}
```

   Where "`ItemsListTable`" in the function `getTagsAccesor().setTagTable("ItemsListTable",gx_table,getGXAppConfig().getTableBuildConfig(),this);` is the name of the `gx:table` tag in the JSP page.

To add the sorting capability to a table (.NET):

1. Generate an ASPX page for the host screen containing the table (using the option generate code).

2. In the generated ASPX page, add two hidden input elements. They will be used to pass the values of the sorted column and determine whether the sorting order should be ascending or descending:

```
<input runat="server" type="hidden" id="sortCol"/>
<input runat="server" type="hidden" id="isAsc"/>
```

3. In the generated ASPX page, add a hidden button that will submit the form when clicking the sort commands:

```
<input runat="server" type="button" id="sortBtn" style="display:none" onserverclick="sort"/>
```

4. In the generated ASPX page, add the following JavaScript function. It will set the selected values into the hidden input fields and submit the page:

```
<script>
function sort(colName,isAscending){
document.getElementById('sortCol').value = colName;
document.getElementById('isAsc').value = isAscending;
document.getElementById('sortBtn').click();
}
</script>
```

5. In the ASPX code of the table, add calls to the JavaScript function in the headers of all columns that should be sorted columns:

```
<th>
Item Name
<img src="images/up.gif" onclick="sort('ItemName','true');"style="cursor:hand"/>
<img src="images/down.gif"
       onclick="sort('ItemName','false');" style="cursor:hand"/> </th>
```

   Where '`ItemName`' in the sort function `sort('ItemName','true');` is the name of the column in the ApplinX table definition. The images for the arrows (`up.gif` and `down.gif`) can be copied from the `Sabrafood /images` folder.

6. In the generated code class of the page (*screenName.aspx.cs*) add the sort function. It will get the values from the request, perform the sort function and build the new sorted table:

```
public void sort(Object sender, EventArgs e) {
GXITable gx_table = gx_session.getScreen().getTables()[0];
        bool isAscending = Boolean.Parse(Request["isAsc"]);
        gx_table.sort(Request["sortCol"],isAscending);
      GXTablesHandler.fillHtmlTableFromGXTable(ItemsListTable,gx_table,this,gx_appConfig.TableBuildConfig);
      }
```

   Where "`ItemsListTable`" in the function `GXTablesHandler.fillHtmlTableFromGXTable(ItemsListTable,gx_table,this,gx_appConfig.TableBuildConfig);` is the name of the table element in the ASPX page.

Sample code can be found in the Composite Demo Application: BrowseProposals.jsp/aspx - contains a screen-based table with sorting capabilities.

Adding the Sorting Capability to a Procedure based Table

It is possible to use the ApplinX Table API to add server-side sorting capabilities to ApplinX Tables in generated Web pages. Usually a Web page displaying a procedure-based table (a table that was collected from several host screens using a navigation path) only collects the table from the host the first time it is loaded, and uses a Web session variable to store ('cache') the table for later use. It is also possible to add a button to refresh the table contents updated from the host. This method saves time and resources, and improves overall performance.

To add a sorting capability to a procedure-based table (using a Web session variable) (JSP):

1. Perform steps 1-4 of the task Adding the Sorting Capability to a Screen-Based Table.
2. In the generated code class of the page (*web-inf\classes\contexts\screenName.java*), override the function `gx_fillTable` in the following way. It will get the table from the session variable or from the host (if it is the first time) and display the table:

```
public void gx_fillTable() throws GXGeneralException{
GXITable gx_table = (GXITable)getSession().getAttribute("ApplinXTableName ");
    if (gx_table == null){
         // executes the procedure and returns a response from which the table can be extracted
         // procedure output - data structure array representing the table
         // Table Name - a string to be assigned to the gx_table name attribute.
         <procedure method Request> req = new CollectAllCustomersRequest();
                        <procedure method Response> res = (<procedure method Response>)getGXSession().executeProcedure(req);
                        GXITable gx_table = GXBindUtil.entityArrayToGXITable(res.get<procedure output>(),"<Table Name>");
            if (gx_table == null){
                return;
            }
         getSession().setAttribute("ApplinXTableName ",gx_table);
    }
// fill the html table with gx_table results
getTagsAccesor().setTagTable("TableTagName",gx_table,getGXAppConfig().getTableBuildConfig(),this);
}
```

   Where `TableTagName` is the name of the gx:table tag in the generated JSP page.

3. Perform step 6 of the task Adding the Sorting Capability to a Screen-Based Table.

To add a sorting capability to a procedure-based table (using a Web session variable) (.NET):

1. Perform steps 1-4 of the task Adding the Sorting Capability to a Screen-Based Table.
2. In the generated code class of the page (*screenName.apsx.cs*), override the function `gx_fillTable` in the following way. It will get the table from the session variable or from the host (if it is the first time) and display the table:

```
public override void gx_fillTable() {
 GXITable gx_table = (GXITable)Session["ApplinXTableName "];
     if (gx_table == null) {
          // executes the procedure and returns a response from which the table can be extracted
         // procedure output - data structure array representing the table
         // Table Name - a string to be assigned to the gx_table name attribute.
         <procedure method Request> req = new CollectAllCustomersRequest();
                        <procedure method Response> res = (<procedure method Response>)getGXSession().executeProcedure(req);
                        GXITable gx_table = GXBindUtil.entityArrayToGXITable(res.get<procedure output>(),"<Table Name>");
           if (gx_table == null){
                return;
           }
           Session["ApplinXTableName"] = gx_table;
// fill the html table with gx_table results
GXTablesHandler.fillHtmlTableFromGXTable(TableTagName,gx_table,gx_appConfig.TableBuildConfig);
}
```

   Where `TableTagName` is the name of the gx:table tag in the generated ASPX page.

3. Perform step 6 of the task Adding the Sorting Capability to a Screen-Based Table.

Changing Table Layout for Instant HTML Pages

ApplinX Tables are created in instant HTML pages using the instant Renderer. By default, the ApplinX host tables, as defined in the repository are displayed as HTML tables. Instant table settings, along with other instant settings should be defined in *GXInstantLogicContext* (JSP)/ *GXInstantLogicWebForm* (.NET) For more information regarding the instant table properties refer to the ApplinX Development API Javadoc (can be found in Start>Program>Software AG ApplinX>Documentation>ApplinX Development API, in the com.sabratec.applinx.presentation.transforms.GXTableConfig class.

Retrieving Values from a Selected Row within a Table

You may want to retrieve a value from a specific row within a table (in a generated Web page). For example the selected row number, or some of the cell values that are in the selected row. Once retrieved, you can use these values for getting further details from another screen, or any other source (such as a database query, within an external web service).

A common scenario is when selecting a table row by activating a server side function that executes a path procedure with a unique ID contained in the HTML table row. In order to return a value from the table to a server side function, it is recommended to use the "selectedKey" hidden control, automatically added by APX framework to all server side tables (runat=server in .NET or gx:table tag in JSP). (See the JSP and .NET table examples below). and then use these values later on in their VB/C#/Java code. (see C#/Java code examples below) The gx_selectKey() function stores the proper row values in the table's hidden control. (see the selectTableRow JavaScript function below)

To retrieve values from a selected row

1. First decide which value/s you wish to retrieve from the selected row. Unless specified otherwise this selected key variable will hold the index number of the row that was selected. However, you can set the framework to store values from the selected row, from a cell/s. This example sets the `customerID` column as the key that will be retrieved.

   Add the following line to your gx_filltable or any other server side function used to fill the table.

   Java Code

   ```
   public void gx_fillTable() throws GXGeneralException
          ........
         // when a row is selected the  customerID value will be stored as the table(Customers) selected key
          getTagsAccesor().addTableKeyColumn("Customers", "CustomerID");
         .......
   }
   ```

   C# code

   ```
   public override void gx_fillTable()
   {
           ........
        // when a row is selected the  customerID value will be stored as the table(Customers) selected key
          GXTablesHandler.addKeyColumn(Customers, "customerID");
          .........
   }
   ```

2. The `gx_selectKey(elem)` function will store the value of the key of the selected row in the selected key variable. The `gx_selectKey` function should be triggered from each row. In the following example, after selecting the row, a server side function (`selectCustomer`) is triggered.

   JSP

   ```
   <script>
           function selectTableRow(elem){
                   gx_selectKey(elem);

     //Triggering a server side function
     gx_postBack("selectCustomer");

   }
   </script>
   .
   ```

```
.
.
<gx:table class="gx_tbl" id="Customers" >
        <tr>
           <th>ID</th>
           <th>Lastname</th>
           <th>Firstname</th>
           <th>Birth date</th>
        </tr>
        <tr style="cursor: pointer;" onclick="selectTableRow(this); ">
           <td id="customerID"></td>
           <td  id="lastName"></td>
           <td id="firstName"></td>
           <td id="birthday"></td>
        </tr>
</table>
```

.NET

```
<script>
        function selectTableRow(elem){
                gx_selectKey(elem);

  //Clicking on a button that executes a server side function
  gx_getElement('SelectCustBtn').click();

}
</script>
.
.
.
<input type="button" id="SelectCustBtn" runat="server"
                  onserverclick="selectCustomer" style="display: none" />
<table runat="server" class="gx_tbl" id="Customers" >
        <tr>
           <th width="68" nowrap="nowrap">ID</th>
           <th width="75" nowrap="nowrap">Lastname</th>
           <th width="69" nowrap="nowrap">Firstname</th>
           <th width="81" nowrap="nowrap">Birth date</th>
        </tr>
        <tr style="cursor: pointer;" onclick="selectTableRow(this); ">
           <td runat="server" id="customerID"></td>
           <td runat="server" id="lastName"></td>
           <td runat="server" id="firstName"></td>
           <td runat="server" align="center" id="birthday"></td>
        </tr>
</table>
```

3. The following server side function retrieves the selected key and adds it as a variable to a path request in order to get the customer's details (`CustomerSelect` path).

   Java Code

```
public void selectCustomer() {
        try {
                GXPathRequest pathRequest = new GXPathRequest("CustomerSelect");
                // Get the Customer table selected key and pass it to the request
                pathRequest.addVariable("Selected_Customer_ID",
                                                        getTagsAccesor().getTableSelectedKey("Customers"));
                GXPathResponse pathResponse = getGXSession().executePath(pathRequest);
                gx_handleHostResponse();
        } catch (GXGeneralException gge) {
                getLogger().errorLog("gx general exception in go to select customer.", gge);
        }
}
```

C# code

```
public void selectCustomer(object sender, EventArgs args)
{
            try
            {
                    GXPathRequest pathRequest = new GXPathRequest("CustomerSelect");
                    // Get the Customer table selected key and pass it to the request
                    pathRequest.addVariable("Selected_Customer_ID",
                                                        GXTablesHandler.getTableSelectedKey(Customers));
                    GXPathResponse pathResponse = gx_session.executePath(pathRequest);
                    gx_handleHostResponse();
            }
            catch (GXGeneralException ex)
            {
                    gx_handleSessionError(ex);
            }
}
```

Refer to the API:

- `gx_selectKey`
- `gx_getSelectedKey`
- `gx_isTableKeySelected`
- `gx_markRow`

Customizing the Table's Display

The display of a table in generated web pages can be customized. For example, in a host table which displays the customer's address, you may want to add an additional conlumn with a link that opens a new browser window with a map of that address.

ApplinX Framework provides three methods that you can use to change the table display:

- `gx_changeTr`: For making changes on the table row level
- `gx_changeTd`: For making changes on the table cell level
- `gx_changeControl`: For making changes on a specific control within a table cell.

In the following example the `gx_changeControl` method is used to apply a dynamic onClick attribute to a hyperlink/Span according to the specific customer address.

JSP

1. Add a column to the table (ensure that you add a suitable header).

```
<td align="center">
     <a target="gmaps" id="Gmaps">
       <img src="../images/map_pin.png"  alt="Map" border="0" />
     </a>
</td>
```

2. Override the `gx_changeControl` method to add an OnClick attribute with a dynamic JavaScript call.

```
public void gx_changeControl(int ColIndex, Element td, Element ctrl,    GXITableRow row)
{
   String ctrlId = ctrl.getAttribute("id");
      if (ctrlId != null && ctrlId.indexOf("Gmaps") >= 0  ){
                String _address = row.getItemContent("HouseNumber").trim();
                _address += " "  + row.getItemContent("Street").trim();
                _address += "," + row.getItemContent("City").trim();
                _address += ","  + row.getItemContent("Country").trim();
           ctrl.setAttribute("onclick","window.open('http://maps.google.com/?q=" + _address + "','gmaps');");
      }
   }
```

.NET

1. Add a column to the table (ensure that you add a suitable header).

```
<td align="center">
     <span runat="server" id="Gmaps">
        <img src="../images/map_pin.png" alt="Map" border="0" />
     </span>
</td>
```

2. Override the `gx_changeControl` method to add an OnClick attribute with a dynamic call.

```
public override void gx_changeControl(int ColIndex, HtmlTableCell td, Control ctrl, GXITableRow row)
{
  string ctrlId = ctrl.ID;
  if (ctrlId != null && ctrlId.IndexOf("Gmaps") >= 0)
  {
      string _address = row.getItemContent("HouseNumber").Trim();
      _address += " " + row.getItemContent("Street").Trim();
      _address += "," + row.getItemContent("City").Trim();
      _address += "," + row.getItemContent("Country").Trim();

      ((HtmlGenericControl)ctrl).Attributes["onclick"] = "window.open('http://maps.google.com/?q=" + _address + "','gmaps');";
  }
  else
  {
       base.gx_changeControl(ColIndex, td, ctrl, row);
  }
}
```

Refer to the CompositeDemo, BrowseCustomers1 page. to the gx_changeTD and gx_changeControl methods.