

ApplinX Exercises

Web Enablement

Version 9.8

June 2015

This document applies to ApplinX Exercises Version 9.8.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2001-2015 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

Document ID: APXE-WEBENABLE-98-20150624

Table of Contents

| | |
|---|----|
| Web Enablement | v |
| 1 Creating a New Web Application | 1 |
| 2 Generating and Customizing Screens | 5 |
| Generate a Screen | 7 |
| Record a Path Procedure | 6 |
| Add a Button which Executes the Path Procedure | 7 |
| Login Page | 8 |
| 3 Handling Screens with Tables/Complex Data | 11 |
| Add a Screen Based Table (via the Screen Editor) | 12 |
| Manually add a Table to a JSP/ASPX Page | 16 |
| Create a Procedure Based Table (from Multiple Screens) | 13 |
| Export a Table to Excel | 15 |
| Things to Take into Consideration when Collecting Data from Multiple Screens | 16 |
| 4 Partial Page Rendering (PPR) | 17 |
| 5 Application Map - Implement Navigating with the Application Map | 19 |
| 6 ApplinX User Exits | 21 |

Web Enablement

Exercise Objectives

In this exercise you will take the ApplinX application you created in the “Instant Demo” segment, and you will take it to the next level, creating a composite web application, demonstrating how to further customize your Host-based web pages using the ApplinX framework.

At the end of the exercise you will know how to:

Create a New Web Application

Generating and Customizing Screens

Handling Screens with Tables

Partial Page Rendering (PPR)

Application Map - Implementing Navigating with the Application Map

Advanced Users: ApplinX User Exits

1 Creating a New Web Application



Create a new Web application.



Recommended reading in ApplinX documentation:

- Getting Started>Creating an ApplinX Web Application

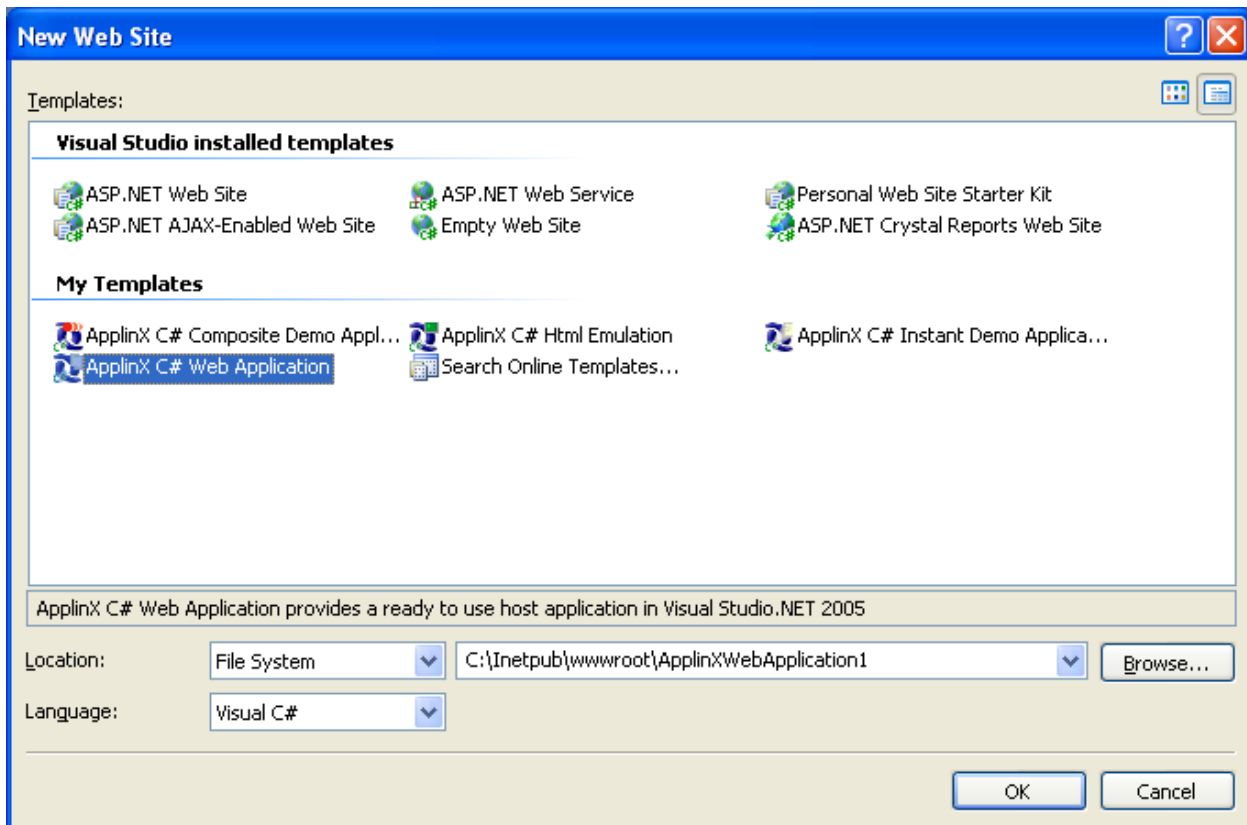


Accompanying movies:

- Creating a New Web Application (JSP)
- Creating a New Web Application (.NET)

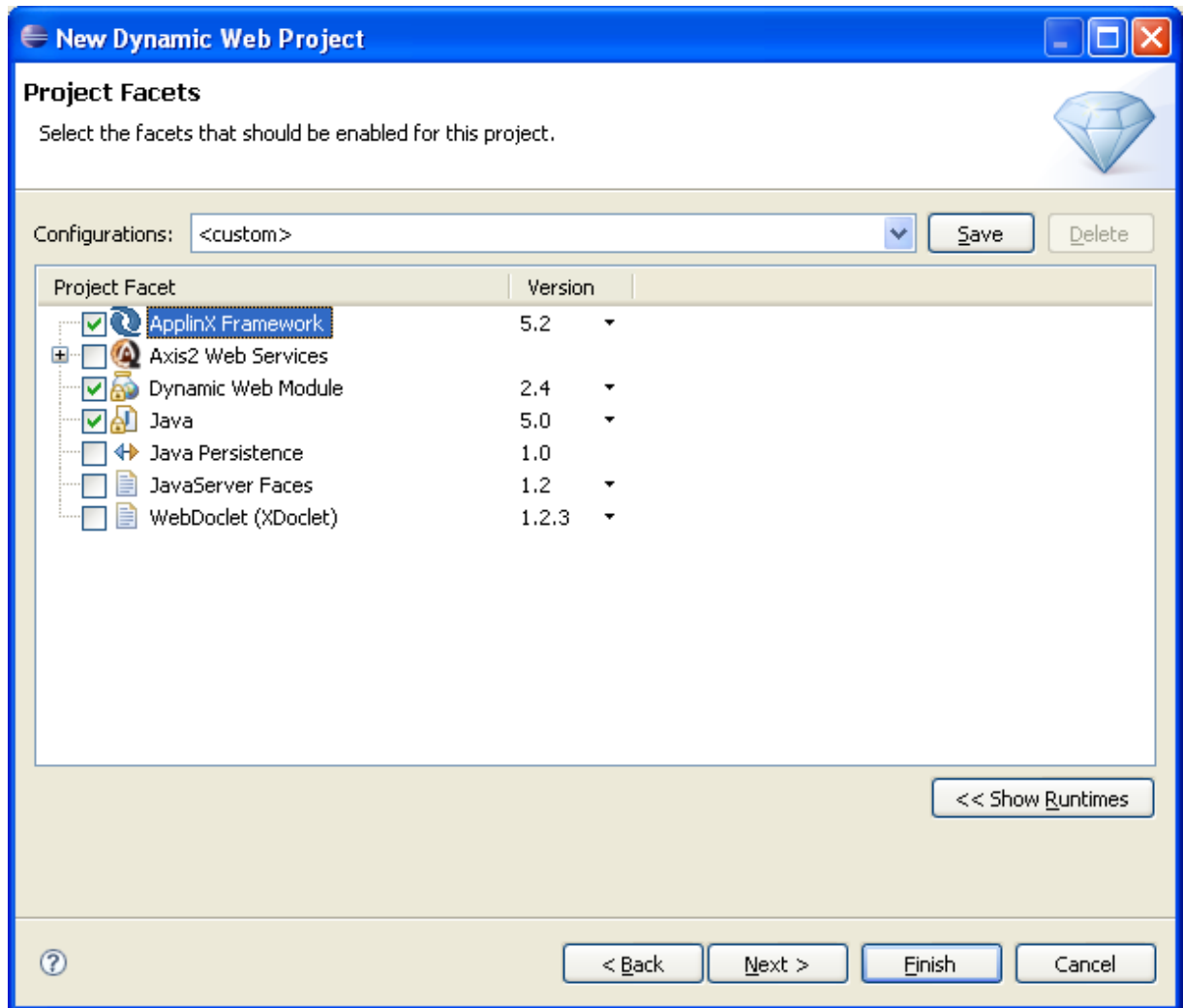
Host screens you didn't generate will be displayed as Instant pages. This methodology lets you take your time developing the web application, since, you don't have to migrate the whole business logic, you can start slowly by generating and customizing the most commonly used screens , working your way down to the least commonly used screens. Processes you didn't get to yet will be displayed as instant pages, looking and functioning as they do in the host.

.NET



JSP/Eclipse Environment

1. Create a new “Dynamic web project”. Name the project. In the project facets screen, check the ApplinX framework box and continue.



2. Run the project and make sure it works properly. If you need to change some of the configurations (Applinx server address, Applinx application name, Instant page font size etc.) you can do so in the Framework configuration page (just click the “configuration” link (this link is used for development use only, make sure you remove it later on)).

2 Generating and Customizing Screens

| | |
|--|---|
| ■ Generate a Screen | 7 |
| ■ Record a Path Procedure | 6 |
| ■ Add a Button which Executes the Path Procedure | 7 |
| ■ Login Page | 8 |

Exercise Objectives

Now that we have created a new project/web application to work with, we will customize host screens. In this exercise we will customize the *Login* screen. When you have completed this exercise, you will know how to:

Generate a Screen



Exercise

1. Generate the Login screen.
2. Open the JSP file and remove unnecessary data and inputs. Apply your own design to the page and leave only essential fields on the screen (username, password, reconnect and message).



Accompanying movies:

- Generating a screen (JSP)
- Generating a screen (.NET)

Record a Path Procedure

When reviewing the logic of the login process, notice that the Splash screen and Menu screen have no substantial use and the user should not have to pass through these screens.



The logical behavior is that the user will go from the Login screen, straight to the InsuranceMenu screen. To implement this, you must create a Path Procedure.



Exercise

1. Record a Login path that navigates from the Login screen to the InsuranceMenu screen:

- Ensure that the path procedure has at least two input variables - user name and password.
 - Ensure that the path procedure has an output variable - errMsg.
2. After recording the path procedure, edit it so that it also handles situations where the wrong user name or password were entered.



Accompanying movies:

- Recording a Path Procedure

Add a Button which Executes the Path Procedure

The login path procedure can be executed by clicking on a button.



Exercise

Create a button that will execute the path procedure with values from the web page and will navigate straight to the InsuranceMenu screen.



Solution steps

1. Add the following to your web page, wherever you see fit:

JSP

```
<gx:input type="button" value="go!!" id="LoginButton" onclick="doLogin" />
```

.NET

```
<input type="button" value="go!!" runat="server" id="LoginButton" ↵  
onclick="doLogin"/>
```

2. Add the following code to your code behind class (java / C#):

Java

```
public void doLogin(){
    try {
        GXPathRequest req = new GXPathRequest("Login");
        String _user = getTagsAccesor().getTagContent("UserID");
        String _pwd = getTagsAccesor().getTagContent("Password");
        req.addVariable("UserID", _user);
        req.addVariable("Password", _pwd);
        GXPathResponse res = getGXSession().executePath(req);
        gx_handleHostResponse();
    } catch (GXGeneralException e) {
        gx_handleSessionError(e);
    }
}
```

.NET

```
public void doLogin(object sender, EventArgs e)
{
    GXPathRequest req = new GXPathRequest("Login");
    req.addVariable("UserID", UserID.Value);
    req.addVariable("Password", Password.Value);
    GXPathResponse res = gx_session.executePath(req);
    gx_handleHostResponse();
}
```

Login Page

In order to display the Login web page as the first page of the web application, without having an actual ApplinX session started, we'll need to make it stand-alone and not dependent on the ApplinX session.

Exercise objectives

The scenario we're interested in is: the Login page comes up, regardless of the ApplinX session. The user enters a user name and password and then the Login page creates a new ApplinX session and executes the Login path. The following steps detail how this is done:



Exercise

1. Change the default inheritance of the Login.java (or cs/vb) class from GXDefaultLogicContext (.NET: GXDefaultLogicWebForm) to GXBasicContext (.NET: GXBasicWebForm). This prevents the page from automatically trying to attach itself to an existing ApplinX session. However, it

also prevents the page from running the `gx_fillForm` method which fills all the form's inputs and labels with host data. We'll have to handle that ourselves.

2. If the host application has other screens before the actual login screen, add these screens to the Login path.
3. Change the `doLogin` method to handle creating a new `ApplinX Session` or attaching to an existing one.

JSP

```
public void doLogin(){
    try {
        String _user = getTagsAccesor().getTagContent("UserID");
        String _pwd = getTagsAccesor().getTagContent("Password");

        GXIConfig config = getGXAppConfig().get_SessionConfig();
        config.setSessionId(_user);
        config.setPassword(_pwd);
        // gx_connect() will attach the Page to an existing ApplinX
        // session that has the provided ID. if none exists
        // a new session will be created.
        gx_connect();

        GXPathRequest req = new GXPathRequest("Login");
        req.addVariable("UserID", _user);
        req.addVariable("Password", _pwd);
        GXPathResponse res = getGXSession().executePath(req);

        gx_handleHostResponse();
    }
    catch (GXGeneralException e) {
        gx_handleSessionError(e);
    }
}
```

.NET

```
public void doLogin(object sender, EventArgs e)
{
    string _user = UserID.Value;
    string _pwd = Password.Value;

    GXIConfig config = gx_appConfig().SessionConfig;
    config.setSessionId(_user);
    config.setPassword(_pwd);
    // gx_connect() will attach the Page to an existing ApplinX
    // session that has the provided ID. if none exists
    // a new session will be created.
```

```
gx_connect();

    GXPathRequest req = new GXPathRequest("Login");
    req.addVariable("UserID", _user);
    req.addVariable("Password", _pwd);
    GXPathResponse res = gx_session.executePath(req);
    gx_handleHostResponse();
}
```


3 Handling Screens with Tables/Complex Data

| | |
|--|----|
| ■ Add a Screen Based Table (via the Screen Editor) | 12 |
| ■ Manually add a Table to a JSP/ASPX Page | 16 |
| ■ Create a Procedure Based Table (from Multiple Screens) | 13 |
| ■ Export a Table to Excel | 15 |
| ■ Things to Take into Consideration when Collecting Data from Multiple Screens | 16 |

Host tabular data can be easily displayed as an HTML table on the web page. Once a table is defined on the host screen, it will be automatically generated to the web page along with the rest of the screen.

In this exercise you will:

Add a Screen Based Table (via the Screen Editor)

Refer to Tables in the Basic Exercises section.



Accompanying movies:

- Creating a Screen Based Table

Manually add a Table to a JSP/ASPX Page

Exercise objectives

Manually create a screen based table without having to generate the screen from scratch. This table can then be used to display a screen based table or can be filled with a data structure array collected with an ApplinX path procedure.



Exercise

Manually add a table to the relevant JSP/ASPX page.

If you already have a web page and don't want to regenerate it, you can manually add an ApplinX table to a web page by just following these simple rules:

- The table ID must match (case sensitive) the table name in the screen Editor.
- Cells in the table that are to contain values from the host screen must either, have an ID or contain an HTML control that has an ID that matches the column (multiple field) name.
- Usually, a table will hold two rows. The first, a header row. And the second, a row that represents the rest of the data rows in the table.

Refer to BrowseCustomers or BrowseProposals in the Composite demo application.

Create a Procedure Based Table (from Multiple Screens)

Procedure based tables are Web page tables that are bound to a path procedure output data, usually collected from several host screens. For example, in a host screen with a table, clicking a host key will scroll down the table.

Exercise objectives

To “collect” the whole table and display it on one web page, as oppose to having the user click a button to get the next set of table records. The following exercise demonstrates how to collect a table from several screens. This isn’t the only way of doing it, but for this common scenario, of a scrollable table, it is the easiest.



Exercise

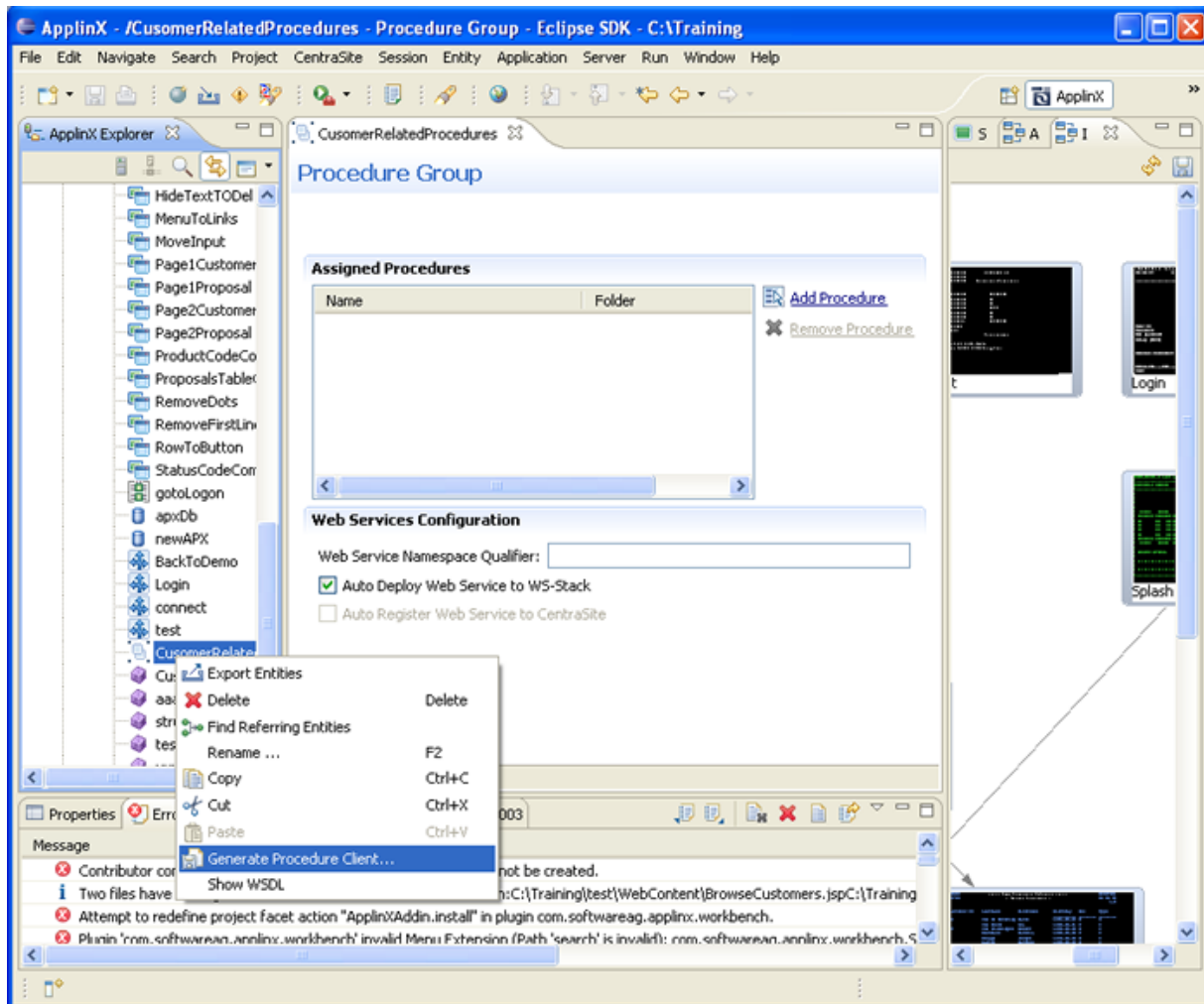
Create a path procedure which collects all customer information as output variables and displays the output on the first customer detail web page, preferably, using tabs.



Solution steps

1. Navigate to a screen with table formatted data (BrowseCustomers, Browseproposals).
2. Identify the screen if has not yet been identified.
3. Map all fields, single fields as well as multiple fields which represent the table’s columns.
4. In the Table tab, create a new table, delete unnecessary columns and change column captions to be more user-friendly.
5. Select Primary key columns. These are typically columns with unique values.
6. Record a Path Procedure that scrolls through the table until it reaches an “End of Data” (or something similar) message on the host screen.
7. Open the path procedure for editing: click the step where the path is scrolled and open its output tab.
8. Select all the required fields from the table entity (hold CTRL and click for multiple select). Then right click the selection and click “New Data Structure”. A dialog box is displayed. Enter a name for the new data structure and click OK. You have just created a Structure that represents one row in the table.
9. Click the “Flow Path” node and open the output tab. Click Add structure and name the variable. Select the newly created data structure from the “type” Combo. Check the “array” checkbox, to make your variable actually represent a table (collection of rows).

10. Add a screen mapper node after the “scroll down” step. Select the relevant screen and then drag the table node onto the newly added variable.
11. Finally, associate the procedure with a procedure group. Create a new procedure group in the repository then add the new procedure to the “Assigned Procedures” box by:
 - Clicking the “Add Procedure” button and selecting the procedure from the dialog.
 - Dragging the procedure from the repository on to the “Assigned Procedures” box.
12. Create a table in the JSP/ASPX page. Follow the rules from the previous section with the exception that Table ID must match the output array parameter and cell IDs must match the Data structure properties names.
13. In order to be able to bind the data from the procedure’s output to our JSP/ASPX table, generate a procedure client from the ApplinX application to the JSP/ASPX project.



14. Uncomment the code in the `gx_fillForm` method and customize it according to your procedure specifications.



Note: This process can be applied not only for binding table data, but also to fill other elements of the web form. For example, if the procedure response has an output variable named “message” and the JSP/ASPX page contains a SPAN tag Named(ID) “message” as well, that span will also be bound to the response data. This way multiple host screens can be displayed on a single web page.



Accompanying movies:

- Collecting Complex Data from Several Screens
- Generating a Procedure Client (JSP)
- Generating a Procedure Client (.NET)

Export a Table to Excel

ApplinX data can be easily exported to an MS Office application, such as an Excel spreadsheet. It is possible to modify the styling and formatting that will be displayed in Excel. This modification can be done in the HTML tags of the XSL file (For example, if you want to add a border to a table, in the excel.xsl file, add to the table tag: <table border=1>).



Exercise

Export a table from a screen which uses a table.



Solution steps

Once we have a GXTable object we can export it to Excel by filling an HTML table:

- The page response content type is set to Excel.
- The page inherits from GXBasicContext/GXBasicWebform. Refer to the Composite demo, Excel page, used in both browseCustomers and browseProposals.



Recommended reading in ApplinX documentation:

- Web Application Development > Page Customization > Exporting an Data to MS Office (Excel, Word)

Things to Take into Consideration when Collecting Data from Multiple Screens

Amount of Data

As we've seen before, it is possible to merge multiple host screens into a single view on the web page, but when should we do this? Here are a few issues we should take into consideration:

- Do the screens have the same context? For example, it is logical to merge customer details spread across a number of host screens into a single web page.
- How many screens are there?

The more host screens you go through, the longer the end user has to wait. Don't try collecting a host table with thousands of rows. Instead, you can let the users use the scroll functionality the host offers or collect 4-5 screens each time the users scrolls.

- What kind of functionality does the host offer in each of these screens, and can it be duplicated/omitted when all screens are merged in to one?

Validating the Web Page Contents

When you merge several host screens into one web page, using JavaScript you can try and perform some of the host's input validations on the client side before the data is submitted to the host or even while the user is typing. Other validations can be performed on the web server side before the data is sent to the host. This is recommended in order to prevent the host.

- Check for required fields.
- Validate date strings, add a calendar Icon and bind it to the date input.
- Check String formats (number, alphanumeric, email, phone etc.)
- Some interaction between input fields.



Refer to Combining Data from Multiple Host Screens in the ApplinX Web Application Development Guide>Working with the Framework section of the ApplinX documentation.

4 Partial Page Rendering (PPR)

It is possible to change only a certain part of the generated web page, when executing server side code, thus, improving ApplinX framework performance by downsizing the amount of data transferred back to the client side.

For example, in a web page which has a screen based table, a user clicks a button that sends [PF8] to the host making the table scroll down on the host side and refilling the web page table with new data. Now, instead of refreshing the whole web page (this will not happen if the table didn't change at all on the host side), ApplinX framework enables sending only the page part containing the table.



Exercise

Create a button that retrieves only the table information from the BrowseProposals screen using the partial page rendering functionality.



Solution steps

Wrap the table with a dynamic server control. Give it a unique ID.

JSP

```
<gx:div id="TablePanel">
<gx:table align="center" cssClass="gx_tbl" id=...
.
.
.
</gx:table>
```

```
</gx:div>
```

```

```

Add the following line of code before you run the server-side function:

```
function scrollDown()  
{  
    gx_updatePagePart('TablePanel');  
    gx_SubmitKey(['PF8']);  
}
```

.NET

```
<span runat="server" id="TablePanel">  
<table runat="server" align="center" cssClass="gx_tbl" id=...  
.  
.  
.  
</table>  
</span>
```

```

```

Add the following line of code before you run the server-side function:

```
function scrollDown()  
{  
    gx_updatePagePart('TablePanel');  
    gx_SubmitKey(['PF8']);  
}
```

Now, whenever you click that link, only the table data will be retrieved from the Web server.

5

Application Map - Implement Navigating with the Application

Map

The Application Map view displays thumbnails of the application screens which the user navigated through, while working with the application. ApplinX saves the navigation through these screens including the steps between each screen such as the action key pressed and the fields sent. The application map can be used in Path Procedures and from the Base Object/Web framework, to navigate to a specific screen, using the `NavigateTo` method.



Exercise

Now that you have created a map, you will use it to navigate from the proposals section to the customer section just by clicking on a button.



Solution steps

Java:

In `GXBasicContext.java` add a server side function that runs a `GXNavigateRequest` ↵ that uses the Application map.

```
public void navigateBut_Click(){
    try{
        GXNavigateRequest navReq = new ↵
GXNavigateRequest(getTagsAccesor().getTagContent("TargetScreen"));
        getGXSession().navigateTo(navReq);
    }catch(GXGeneralException err){
        gx_handleSessionError(err);
    }
    finally{
        try{
```

```
        gx_handleHostResponse();
    }catch(GXGeneralException err2){
        gx_handleSessionError(err2);
    }
}
```

.NET:

In template.master.cs add a server side function that runs a GXNavigateRequest that uses the Application map.

```
public void navigateBut_Click(object sender, EventArgs args)
{
    try
    {
        GXNavigateRequest navReq = new GXNavigateRequest(TargetScreen.Value);
        gx_page.gx_session.navigateTo(navReq);
    }
    catch (GXGeneralException err)
    {
        // take us to the screen the host is on
        gx_page.gx_handleHostResponse();
    }
    finally
    {
        try
        {
            gx_page.gx_handleHostResponse();
        }
        catch (GXGeneralException err2)
        {
            ///handle error
            gx_page.gx_handleSessionError(err2);
        }
    }
}
```



Recommended reading: Designing and Developing an Application>ApplinX Entities>Application Map.

6 ApplinX User Exits

Exercise objectives

The object of this exercise is to get to know how to use ApplinX User Exits. ApplinX framework includes various user exits, enabling the developer to intervene with framework events and methods both on the server-side (JAVA/CS/VB) and on the client side (JavaScript).



Recommended reading in ApplinX documentation:

- ApplinX Development API>Server Side API>General> User Exits
- ApplinX Development API>Client Side> User Exits



Exercise

In a screen that has a table, merge several related columns into a single column and display this in the web page table.

Related columns may be, for example, address related columns, such as country, city and zipcode or street and house number.



Solution steps

Use the `gx_changeTd` method to change the default appearance of cells within tables. Refer to BrowseCutomers1 Page in the CompositeDemo web application, and pay close attention to the use of the `gx_changeTd` method.

