ʃ software ᴬᴳ

ApplinX User Guide

Web Application Development

Version 9.6

April 2014

webMethods ApplinX

This document applies to ApplinX Version 9.6.

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

Table of Contents

I Introduction and Concepts

1 What are ApplinX Framework for JSP and ApplinX Framework for .NET Web Applications

The ApplinX Framework for ASP .NET and ApplinX Framework for JSP are fully functional Web applications. The ASP .NET Framework runs under the .NET Framework (CLR) and IIS and the ApplinX Framework for JSP runs under various application servers (refer to Recommended Software). The ApplinX Frameworks use the ApplinX Base Object, to communicate with the ApplinX Server.

Immediately upon installing ApplinX Framework for ASP.NET or ApplinX Framework for JSP, you possess a fully functional Web application. A completely structured Web application is prepared, and there is no need to add even one line of code.

These frameworks serve as excellent platforms for advanced development, as you combine the ApplinX entities and functions to your Web development platform.

2   Development Methods

The ApplinX Framework contains three development methods:

Host Driven Development

In this development method the Web application is developed in a framework driven manner, which means all activity is performed by the ApplinX Web application and you have the ability to make changes within the framework flow.

The use of this methodology is accomplished when inheriting from *GXDefaultLogicContext (JSP)/GXDefaultLogicWebForm (.NET)* , which includes host keys' handling, and in general, an automatic workflow with the ability to make changes using user exits.

Instant Driven Development

In this development mode, most host screens remain as Instant HTML pages, meaning they are generated on the fly by ApplinX Framework, without generating specific Web pages (JSP or ASPX) for individual screens. Thus, the development process is quicker and simpler, and maintenance efforts are significantly reduced.

Customization of the instant pages is done using Instant Transformations and using Screen Groups. Often, there isn't even a need to identify individual screens. Server transformations defined using the Transformation wizard in ApplinX Designer will be displayed in the instant pages. The page used in the kind of development is *GXInstantLogicContext (JSP) /GXInstantLogicWebForm (.NET)* , which contains Instant Transformations registration.

Server-Side Event Driven Development

The ApplinX framework exposes another methodology for development. You may use this methodology by changing inheritances for the project pages to *GXBasicContext (JSP)/GXBasicWebForm (.NET)*. This methodology is accomplished by using server-side buttons and the use of the framework API's as "building blocks" (gx functions).

ApplinX Procedures Development: The ApplinX framework allows you to minimize the logic in the application classes. In this case, the framework's purpose is to serve as a controller/viewer, using Web pages with dynamic tags/controls. Most of the logic development is performed using flow procedures, which accept and return simple parameters/data structures. After developing the flow procedure, the ApplinX Add-in generates procedure client classes (which should not be modified) and a Web page (you can customize to suit your needs), which is bound, bi-directionally into the Web page using the code class, and binding functionality. Refer to **Working with Procedures**.

3   **ApplinX Frameworks' Key Features**

ApplinX Framework for JSP is based on Java and JSP frameworks standards. ApplinX Framework for .NET is built upon ASP.NET and is available both for C# and for VB.NET. The frameworks provide the following features:

Screen Access API using ApplinX Base Object

The Base Object supports retrieval and manipulation of host entities such as screens, fields and so on as objects, as well as communication with the host using requests (for executing a navigation path or sending a host key) and error handling using exceptions. Refer to Base Object.

Separation of logic

Separation of logic and creation of reusable, self-contained components. Each page contains its own logic, and common logic is encapsulated in components or "building blocks".

Code Transformations

In addition to the server transformations defined using the Transformations wizard, the instant component exposes an API which enables using code transformations. These code transformations allow full flexibility to manipulate the HTML output and implement functionality not provided when using the transformation wizard.

"Building Blocks"

Common framework functionality is divided into easy-to-use methods or "building blocks". These building blocks are customizable and can be modified and overridden by the developer.

Server HTML Controls

The framework uses standard HTML controls/tags as server controls, allowing straight forward binding to standard HTML. In JSP the framework uses gx HTML tags. In .NET the framework uses HTML controls.

Visual designers

Using eclipse 3.3 (for JSP framework) or Visual Studio/Express Edition (for .NET framework) provides a visual designer for your application Web pages.

Separation between data, logic and view.

Pages can be designed by graphical designers or Web designers and easily bound to ApplinX elements. The pages contain only presentation and visual display code, while code implementing logic and data is written in the "code behind" of the page.

Developer's intervention points in the default workflow of the framework.

ApplinX offers several "user exits" that can be customized for the entire project or per specific pages.

Direct navigation menus

Direct navigation menus using ApplinX Navigation Maps. Web menus supporting direct navigation to screens in the host application can be easily implemented using ApplinX Maps.

Built-in support for host windows

"Modal" host windows can be displayed as Web pop-up windows using a built-in, fully customizable framework mechanism.

Host Tables

ApplinX Tables can be easily bound to .NET HTML Table or .NET DataGrid control (.NET) or to an HTML table tag (JSP) and displayed in any desired format (such as graphs) or exported to external tools such as Microsoft Excel.

Integration with Visual Studio .NET and Eclipse.

Easy creation of new ApplinX Web applications. ApplinX Add-In generates Procedure Clients for ApplinX Procedures.

Clustering support

Both JSP and .NET frameworks support clustering in the relevant environments.

Organization and modularity

Integration with ApplinX folders structure for hierarchically organizing Web pages.

JSP Tag Library

Included with the new JSP Framework is a ready-to-use HTML tag library. The tag library is easy to learn since it uses standard HTML tags and attributes. The tag library contains dynamic content

and attributes, while not containing any <%%> code within. All the tags, dynamic content and attributes are managed by the TagsAccessor object, which you can access in the context Java class. You can easily expand the tag library for new tags and new tags' attributes.

Working in Software AG's Designer to develop an ApplinX Framework for JSP Project

It is recommended to develop the JSP framework using Software AG's Designer. Advantages:

- The Web server (e.g. Tomcat) is run within the IDE.
- Embedded Java compiler for the context Java classes
- Auto-complete code based editors (such as JSP and JAVA)
- Debugging
- and more...

4   **Class Hierarchy**

Class Hierarchy - .NET Framework

```
System.Web.UI.Page
            Com.sabratec.dotnet.web.GXScreenBasedWebForm
                 GXBasicWebForm
                       GXDefaultLogicWebForm
                             or
                       GXInstantLogicWebForm
                              YOUR_PAGE.aspx.cs/vb
                                     YOUR_PAGE.aspx
```

System.Web.UI.Page is the top-level ASP.NET class. All aspx pages should inherit from it directly/indirectly.

This class contains the life cycle mentioned in The Framework Lifecycle.

GXScreenBasedWebForm is an ApplinX ASP.NET framework class that contains all the objects and functionality needs for the ApplinX framework.

> **Note:** It does not perform any logic when inheriting from this class. It only provides the functionality.

GXBasicWebForm is the top-level class in your project. All pages in your ApplinX ASP.NET framework project should inherit from this class directly or indirectly. It allows you to declare configuration for all project pages, since all the pages inherit from it. In this class you should add project level code (events/user exits, overriding ApplinX framework gx functions). This class does not perform any logic.

GXDefaultLogicWebForm is a class that performs the default ApplinX framework logic: attaching/connecting, check synchronization, sending the fields when submitted by keyboard keys, jumping to the next page (instant/generated), filling the page fields and error handling.

All ApplinX JavaScript functionality is added to a page when inheriting from this page. Refer to Host Driven Development method.

GXInstantLogicWebForm is a class that contains all configuration parameters of Instant HTML rendering, and the registration of instant transformations.

Your Page.aspx.cs(vb) is the code behind a specific Web page. You should add logic only for the specific page. Should inherit from GXBasicWebForm/GXDefaultLogicWebForm.

Your Page.aspx contains the design of a Web page.

> **Note:** Strong names are used with ApplinX .Net framework assemblies.

Class Hierarchy - JSP Framework

```
com.sabratec.j2ee.framework.web.GXWebPageContext
                            ↵
com.sabratec.applinx.j2ee.framework.web.GXScreenBasedJspContext
                                      GXBasicContext
                                              GXDefaultLogicContext
                                                          or
                                              GXInstantLogicContext
                                                      <YOUR_PAGE>.java
                                                              ↵
(contains)<YOUR_PAGE>.jsp
```

com.sabratec.j2ee.framework.web.GXWebPageContext is the top-level context class. All context java classes should extend from it directly/indirectly. It contains access to all the JSP objects: request, response, session, application, Writer and additional Sabratec JSP objects (non host related): logger, tagsAccessor and window.

All of the objects can be accessed by the get<OBJECT_NAME> method.

This class contains the life cycle mentioned in The Framework Lifecycle.

Each JSP page contains a reference to a context class which can be referred as <YOUR_PAGE>.java

com.sabratec.applinx.j2ee.framework.web.GXScreenBasedJspContext is an ApplinX Framework for JSP class that contains all the objects and functionality needs for the ApplinX Framework. It contains access to GXWebAppConfig and GXSession, using the get<OBJECT_NAME> method, and access to all the framework building blocks ("gx_" function).

> **Note:** It does not perform any logic using this context class. It only provides the functionality.

GXBasicContext is the top-level context class in your project. All java context classes in your JSP Web application project should extend from this class directly or indirectly. It allows you to declare general code and user exits for all project pages, since all the context classes extend from it. In this class you should add project level code (events/user exits, overriding ApplinX framework gx functions). This class does not perform any logic.

GXDefaultLogicContext is a class that performs the default ApplinX framework logic: attaching/connecting, check synchronization, sending the fields when submitted by keyboard keys, jumping to the next page (instant/generated), filling the page fields and error handling.

All ApplinX JavaScript functionality is added to a page when using this context. Refer to Host Driven Development method.

<Your Page>.java is the context class for a specific Web page. You should add logic only for the specific page. Should extend from GXBasicContext/GXDefaultLogicContext.

<Your Page>.jsp contains the design of a Web page with static HTML and tags from the HTML tags library. The JSP framework should declare in the gx:html root tag, which is the context class.

5 **The Framework's Basic Life cycle**

The framework is built upon a concept of a page life cycle, which is a general concept and not related to ApplinX host applications directly. The life cycle of a page is declared by the top context class:

JSP: com.sabratec.j2ee.framework.web.GXWebPageContext.

.NET: System.Web.UI.Page

Each page decides which is its class by declaring the root node: `<gx:html gx_context="<CONTEXT_CLASS>">` and closing it with `</gx:html>` as the end tag.

Each dynamic tag should be set with a prefix of "gx:" for example: `<gx:input id="CustomerId"/>`. The `<CONTEXT_CLASS>` is initialized and starts the page Life cycle:

- **gx_onInit (JSP) /OnInit (.NET) :**

    Used to initialize page and/or project settings, and register to events. For example setting the ApplinX configuration and registering classes to user exits.

- **gx_onLoad (JSP) /OnLoad (.NET) :**

    Used to fill the tags with run-time data from data sources. Host data will be used in lower inheritance level to fill the tags with host data. Page.IsPostBack (.NET)/ gx_isPostBack (JSP) can determine if the page is first called, or submitted back to itself.

- **Post back:**

    Performed automatically by the framework. Each page is submitted to itself using server side buttons/links:

JSP:

```
<gx:input id= myBtn  onserverclick= <FUNC_NAME> />
<gx:a id= myLinkBtn onserverclick= <FUNC_NAME> >Send</a>
```

.NET:

```
<input id="myBtn" runat="server" onserverclick="< FUNC_NAME>"/>
<a id="myLinkBtn" runat="server" onserverclick="<FUNC_NAME>">Send</a> ↵
```

The `<FUNC_NAME>` will be activated automatically upon a user click, and it will be used to perform update actions, re-queries or jumping to another page using redirect. In the context class the developer should hold a function in the format:

```
Public void <SERVER_FUNC>(){
     // code in response to the server click event.
} ↵
```

Can be also fired using JavaScript using: `gx_postBack("<SERVER_FUNC>");`

```
public void <SERVER_FUNC>(Object sender,EventArgs e){
     // code in response to the server click event.
} ↵
```

In JSP the event can also be fired using JavaScript: `gx_postBack("<SERVER_FUNC>");`

▪ **gx_preRender (JSP) / PreRender event (.NET):**

Used to add logic after any post back event for example: refilling the page.

▪ **Controls/Tags rendering:**

The dynamic tags are rendered with runtime content and attributes manipulation performed by the tagsAccessor in the previous stages of the life cycle.

▪ **gx_onUnload(JSP)/OnUnload (.NET):**

Occurs when the page ends. Used to release any relevant resources.

▪ **gx_onError (JSP)/Error event (.NET):**

Used for capturing errors. Any known thrown or runtime error is captured and can be analyzed by the developer.

6 **Web Page Structure**

The top URL of the framework is z_container.jsp /aspx which is a servlet/HTTP handler that creates a frameset with the relevant page components.

Framesets are used in the framework as follows:

■ ApplinX framework contains a huge amount of JavaScript. Using a frameset, the JavaScript files are loaded only once, within the frameset, when the Web session starts. Further more, when using HTTPS in the Web application, JavaScripts are not cached in the browser and this caused slower response time. As the JavaScript files are loaded only once when using framesets, this performance problem no longer exists.

  The frameset listens to the onLoad event of the displayed frame, and listens to any event occurring in the displayed frame (keydown, focus, etc.). Refer to Configuring your Framework and to the JavaScript Logger Engine.

■ The frameset contains additional features (each feature is in a different frame): JavaScript logger, Printlet (optional) and ActiveX (optional).

■ The frameset contains two frames for displaying Web pages in the following cases:

  ■ Usage of modal windows – Opens a window when there is one in the host, loaded from the inactive frame.

  ■ Natural data transfer – Used for opening upload/download windows, and for displaying upload/download status.

  ■ Prevent page refresh effect – The inactive frame becomes the active frame, only after the page is loaded.

■ The two frames constantly switch from active to inactive between themselves, while submitting the data from one to the other.
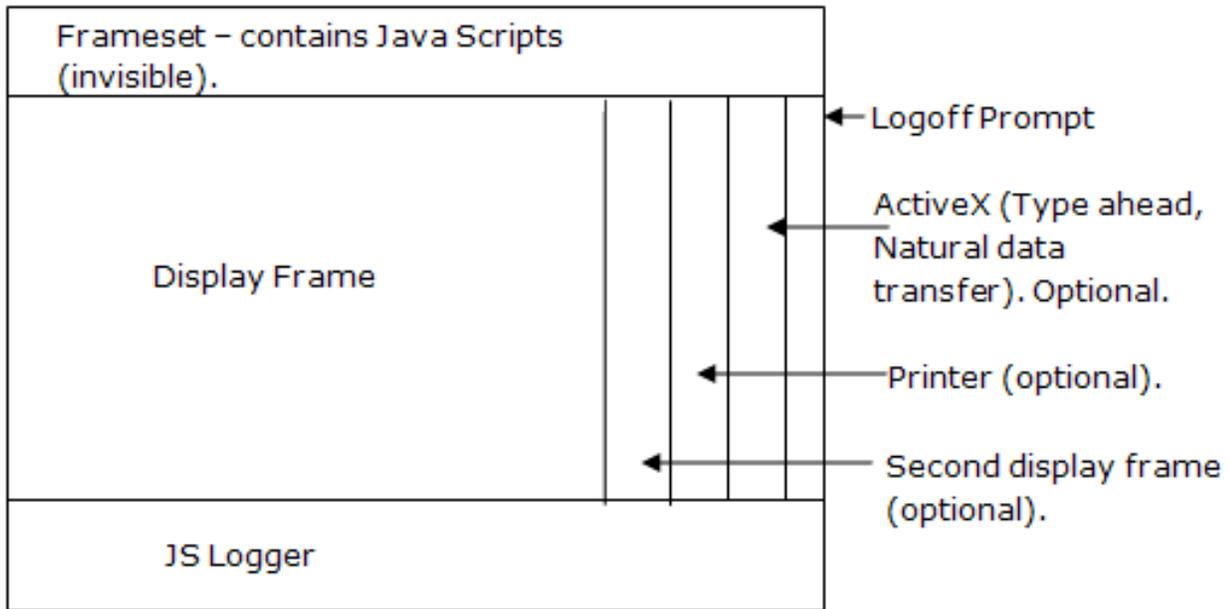
Modal windows also work with a frameset, but only with two display frames (all the other frames are only used in the main browser).

Refer to Configuring your Web Application and to Web Application Configuration Parameters.

7 Pop-Up Manager

One of the biggest problems in session based host-to-Web applications is managing pop-up windows since Web applications window events are not related to server-side code. The Pop-up Window Manager's main aim is to solve this problem, by providing a clear API for managing pop-up Web windows on the server and client-side.

This feature provides the following solutions:

- Two objects (`gx_window and gx_window.opener`), with the same methods on the server and client-side, for opening, closing, refreshing, updating etc. Web windows.

- Natural integration within the ApplinX framework, so when a host window is recognized, a proportionate matching Web window is opened.

- The window object is independent from the ApplinX JavaScript engine.

- Different pages for instant windows (*instantWin.jsp* (JSP) /*instantWin.aspx* (.NET)), for different designs (without page templates for example).

- Provides server-side events for client-side events, as follows:

  - Server-side code event for the pop-up window close event, to allow synchronization with the host.

  - Server-side code event for main window close event, to allow closing the host session.

  - Main window refresh event called when the pop-up window is closed (can be canceled).

- ApplinX framework provides User Exits for canceling Web window openings (when you are not interested in displaying a host window as a Web window), or changing the default opening window size.

- ApplinX framework provides User Exits (`gx_changeNextForm`) for opening a non-host window as a Web window.

- Pop-up windows are modal, so the user cannot perform any action on the main window, which may cause synchronization problems.

■ Allows you to perform updates from a pop-up window to the main window using client-side code, server-side code or main window refreshing.

■ Solves the browser "flickering" affect. After pressing a host key, causing a minor change to the screen, the screen will display this minor change, without seemingly reloading the entire page (i.e. without the user seeing the "flicker" affect).

This feature uses a frameset file (*gx_container.jsp* (JSP)/*gx_container.aspx* (.NET)) to perform submits from the left part of the frameset to the right part, and back. This way, first the Submit is performed, and then if the host contains a window, it will be opened as a Web window. The gx_window represents the pop-up windows' API. Each gx_window object has access to its opener window which is also a gx_window object, and this way a pop-up window can access, modify, etc. its opener window both on the client and server-side. For example, gx_window.opener.refreshPage(), will refresh the opening page if the current window is an opener.

8   Instant Solution

The instant solution offers a methodology for developing Web enabling applications by providing the instant API and a variety of features that accelerate development.

Instant Web Application Development Methodology

Following are some theoretical guidelines regarding development of ApplinX Instant Web Applications, using Instant Transformations and Screen Groups. For practical guidelines and tasks, refer to Getting Started with an ApplinX Instant Web Application.

- A Transformation (or Transform) is a code class (written in Java for the JSP framework and in C# or VB.NET for the .NET framework), that is executed on all host screens or on a specific group of screens (i.e. an ApplinX Screen Group) and modifies the rendered instant HTML page for the current host screen it's executed on.

  Transformations can be used to:

  - **Manipulate individual host fields:**

    For example, display a title field in an underlined, blue font or display an image instead of a description field.

  - **Manipulate entire screens:**

    For example, convert menu screens to a list of hyperlinks.

  - **Manipulate host characters:**

    For example, remove all unnecessary dots (.) or dashes (-) from the screen.

- Using transformations: Transformations should be used to manipulate all screens in the host application or groups of screens (using Screen Groups). To manipulate the contents of specific, individual screens, the recommended methodology is to identify the screen(s), generate a Web page for it (JSP or ASPX) and modify the generated page contents as necessary. Using a transformation for an individual screen is more complicated and unnatural, and therefore not recommended.

- Identifying screens: As a result, in order to use transformations it is not necessary to identify individual screens, but only screen groups (if needed), or not identify anything. This way, development effort is reduced and future maintenance becomes simpler.

- Mapping Fields to Screen Groups: The recommended methodology is to map all host fields that are common to several screens to a screen group that will be associated (implicitly or explicitly) with these screens. For example, an error message and title can be mapped to a screen group that will apply to all screens (or to specific screens), and should not be mapped repeatedly to individual screens. (Note that it is possible to override the mapping definition of the Screen Group in individual screens in which the field appears in a different position). If such mapping is not possible, refer to the fields according to their position or a unique identifier (such as preceding text).

- Manipulating fields: Once fields are mapped to a screen group, it is possible to write Transformations manipulating them.

- Manipulating entire screens: Once screen groups have been defined for the relevant screens, it is possible to write a transformation that is executed for these screen groups. For example, in order to convert a menu screen into a list of hyperlinks, create a Screen Group that will implicitly apply to all menu screens in the host application, and then create a transformation that will be executed for the Menu screen group and will do the required conversion. Note that it is not recommended to create transformations for individual screens, but rather generate Web pages for them.

- Manipulate host characters: Manipulation of host characters usually applies to all screens in the host application, so the recommended methodology is to write a transformation that will manipulate the characters as desired and will be registered for all host screens. When not using the methodology above, it is also possible to create a screen group for the relevant screens and register the transformation only for the screen group.

Instant Features

Absolute HTML Rendering

The instant methodology renders the HTML tags in absolute terms. Rendering the HTML tags in absolute terms prevents indentation problems when trying to add or change elements or style sheets such as GUI elements, host windows and different fonts. In this version, each tag is rendered with corresponding top/left attributes which places the tag in the browser, in an exact position. The main advantage of this approach is that changing/adding tags/style sheets doesn't affect the neighboring tags of the modified tag. In addition, rendering in absolute terms saves the need to render empty spaces and only the relevant fields with content are rendered and not the entire screen. This significantly improves bandwidth performance since the HTML is much smaller.

Instant Rendering API

The instant component opens the instant solution as an API. It provides you the abilities to register as many transformations (which are standard code classes) as you require. These transformations can manipulate the HTML output by adding, removing, replacing and changing the screen tag model using a rich library of tags and an easy-to-use API.

9 Class Hierarchy - Working with Procedures

JSP

```
com.sabratec.j2ee.framework.web.GXWebPageContext
com.sabratec.applinx.j2ee.framework.web.GXAbstractProceduresPageContext
<YOUR_PAGE>.java
(^ contains)<YOUR_PAGE>.jsp
```

com.sabratec.j2ee.framework.web.GXWebPageContext : refer to Class Hierarchy—Framework description.

com.sabratec.applinx.j2ee.framework.web.GXAbstractProceduresPageContext is an ApplinX Framework for JSP procedure class that contains all the objects and functionality needed for working with procedures. It contains access to GXWebAppConfig, using the get<OBJECT_NAME> method, and access to all the framework building blocks ("gx_" function), that can bind the request and response of a procedure into a JSP page.

Note: It does not perform any logic using this class—it only provides the functionality.

<Your Page>.java Contains logic that executes the relevant procedures, fills the page with the procedure response, and responds to server-side events in order to send user typed data to procedures, and jump to the next relevant page.

<Your Page>.jsp: refer to Class Hierarchy—Framework description. Each JSP page contains a reference to a context class which can be referred as <YOUR_PAGE>.java.

.NET

```
System.Web.UI.Page
com.sabratec.dotnet.framework.web.GXAbstractProceduresWebForm
<YOUR_PAGE>.cs
<YOUR_PAGE>.aspx
```

System.Web.UI.Page is the top-level ASP.NET class. All aspx pages should inherit from it directly/indirectly. This class contains the life cycle mentioned in The Framework Lifecycle.

com.sabratec.dotnet..framework.web.GXAbstractProceduresWebForm is an ApplinX ASPX procedure class that contains all the objects and functionality needed for working with procedures. It contains access to GXWebAppConfig, and access to all the framework building blocks ("gx_" function), that can bind the request and response of a procedure into a JSP page.

Note: It does not perform any logic using this class—it only provides the functionality.

<Your Page>.cs: Contains logic that executes the relevant procedures, fills the page with the procedure response, and responds to server-side events in order to send user typed data to procedures, and jump to the next relevant page.

<Your Page>.aspx: refer to Class Hierarchy—Framework description.

GXAbstractProceduresPageContext / GXAbstractProceduresWebForm

Package: `com.sabratec.applinx.j2ee.framework.web`

Contains gx "building blocks" that automate the process of working with procedures. All JSP procedure pages should inherit from this class.

gx_fillTable(GXISelfSerializable[ ] entities)

Accepts a request object array returned from a procedure, converts it into a GXITable entity, and fills the table tag with data according to the gx tags ID. Refer to Working with Tables for more details.

Note: All types of Applinx Data Structure attributes are supported by gx_fillTable.

gx_fillForm(GXISelfSerializable entity)

Accepts a response object/internal entity (of the response object) returned from a procedure, and fills the gx tags with data according to the entity field and the gx tags Id's.

Note: Binding is performed only for the top-level fields of the passed entity.

If the root level of the passed entity contains an entity array, it is also performs gx_fillTable.

Note: All types of Applinx Data Structure attributes are supported by gx_fillForm.

gx_prepareEntity(GXISelfSerializable entity)

Accepts an initialized request entity (new) / internal entity (of the request object), and fills it with data from the JSP page. Only the root of the passed entity level is filled.

Note: All types of Applinx Data Structure attributes are supported by gx_prepareEntity.

II   Working with ApplinX Frameworks

Deploying an ApplinX Web Application (.NET)

Disconnecting the Host Session Correctly

Troubleshooting your Framework	Performance Monitoring

JavaScript Logger Engine

Investigating the Web Application's Code

10 Configuring your Web Application

ApplinX provides a Web based configuration editor where you can configure framework parameters. These parameters are saved in config/gx_appConfig.xml file. Refer to Web Application Configuration Parameters for a list of the parameters, and to the help in the Configuration Editor.

> **Note:** These parameters can be manually configured in the gx_appConfig.xml file.

▶ **To configure the framework:**

1   Open a new browser and run your Web application.

2   Click on the Configuration link. The Configuration Editor will be displayed.

The left side of the window displays the nodes. Clicking on a plus sign expands the node and displays all the parameters in this node. Click on a parameter/node to display the relevant Javadoc on the right side of the window.
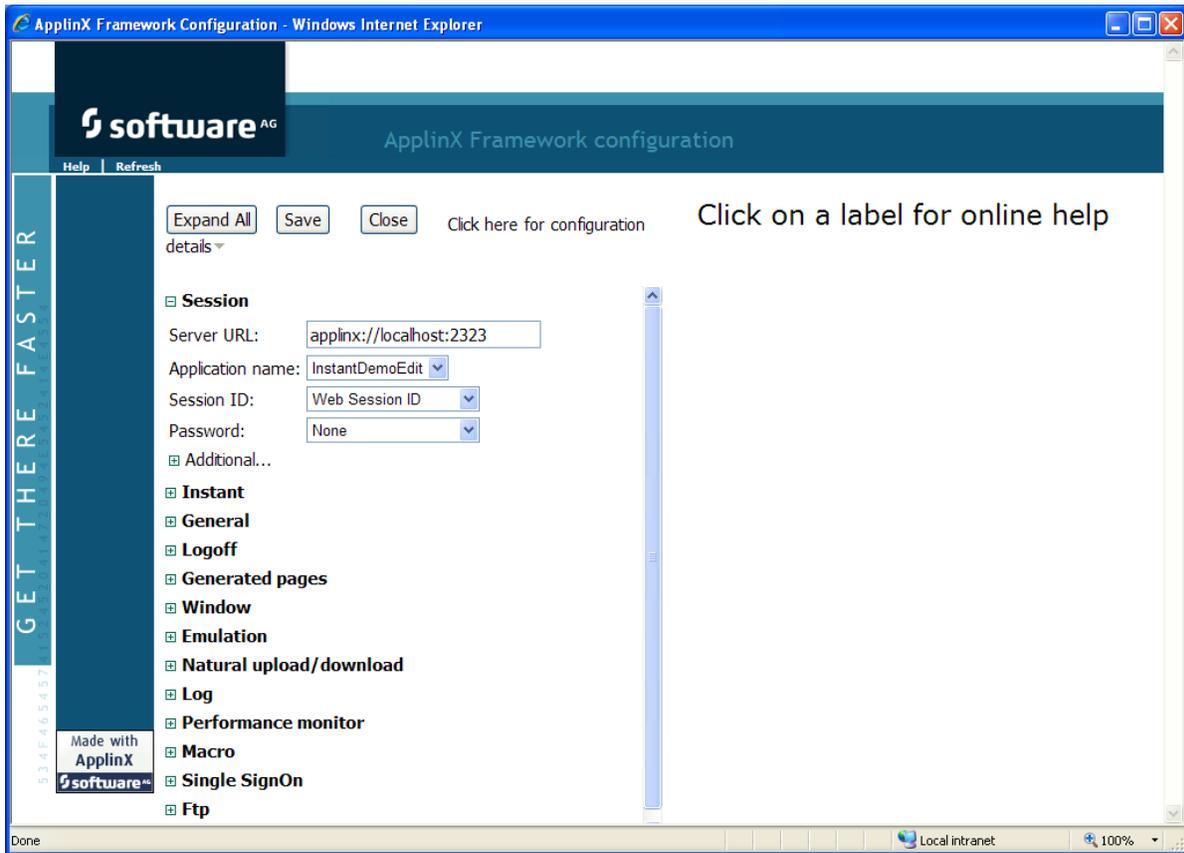
3   Expand and collapse the various nodes as necessary to change the framework parameters.

4   Click Save to save your changes.

5   Click Close to return to the Web application.

> **Note:** When using Eclipse, in order to implement the framework configuration changes, in both the design and runtime Eclipse environments, select the config\gx_appConfig.xml configuration file in your project folder.

> **Note:** In IE8 and above, when opening a new tab in the browser and navigating to your Web application, the same ApplinX session will be used in both tabs.

Troubleshooting .Net Configuration

In .Net, when attempting to save the changes made in the Configuration Editor, you may receive an error message: **Access denied**. In order to resolve this problem, follow these instructions:

1. Click on **Click here for configuration details**.

2. Open Windows Explorer and right-click on the *config* directory in your Web application (*<YourWebApp>\config*).

3. Select **Sharing and Security...** and then select the **Security** tab.

4. Click **Add...**.

5. If you are working with a network/active directory, you need to change the location and select the server name/computer name.

6. Add the IIS process user, typically called ASPNET, to the list of object names that it is possible to select. Click **Check Names** to locate the defined user. This is relevant for IIS 5.1. For any other IIS, refer to **http://support.microsoft.com/kb/895967** for the correct user name and permissions.

7. Click **OK.**

8. Add **Modify** and **Write** permissions to the folder where the configuration is located. Click **OK.**

9. Restart IIS.

10. In the Framework Configuration Editor, re-save your configuration.

11 General Application Customization

Customizing the Default Template

Every ApplinX Web application contains a template file which is used by all the pages. It is possible to override parts of the default template by using placeholders. These placeholders are defined in the template and can be overridden with customized content on generated pages and generate screen groups. Using placeholders in the template provides flexibility on the overall look and feel of individual pages. For example, if the template supplies the users with links on the left side of the page and on a specific page, it is required that these links be removed, the template's default content can be overridden for that page section, with customized content for that particular page. Refer to the Composite demo, template.jsp/template.master and login.jsp/aspx.

▶ **To create a custom template for a screen/screen group (JSP):**

1   Generate a Web page for the screen/screen group (refer to Generating a Framework Page for a Screen/Screen Group).

2   For each template part that you would like to design for this screen/screen group, ensure that the area that is to be overridden is enclosed by a `gx:placeholder` tag. For example, if you would like to change the page header, implement the following in the *template.jsp* file.

```
<gx:placeholder id="<TemplatePlaceHolderID>">
    .
    .
        <your page header JSP code>
    .
    .
</gx:placeholder>
```

3   Server side functions referred by the template.jsp file will have to be placed in the GXBasic-Context.java file.

4   In the jsp file that was generated for the screen/screen group (*generatedpage.jsp*), override a template placeholder by adding your content and enclosing it within a gx:content tag, setting the attribute `ContentPlaceHolderID` with the ID of the template place holder previously defined.

```
<gx:content ContentPlaceHolderID="<TemplatePlaceHolderID>">
    .
    .
                <YOUR CONTENT HERE>
    .
    .
</gx:content>
```

▶ **To create a custom template for a screen/screen group (.NET):**

1   Generate a Web page for the screen/screen group (refer to Generating a Framework Page for a Screen/Screen Group).

2   For each template part that you would like to design for this screen/screen group, ensure that the area that is to be overridden is enclosed by a `asp:contentplaceholder` tag. For example, if you would like to change the page header, implement the following in the *template.master* file.

```
<asp:contentplaceholder id="<TemplatePlaceHolderID>" runat="server" >
    .
    .
       Links section
    .
    .
</asp:contentplaceholder>
```

3   Server side functions referred by the template.master file will have to be placed in the *template.master.cs/vb* file.

4   In the aspx file that was generated for the screen/screen group (*generatedpage.aspx*), override a template placeholder by adding your content and enclosing it within a asp:Content tag, setting the attribute `ContentPlaceHolderID` with the ID of the template place holder previously defined.

```
<asp:Content ID="<tagID>" ContentPlaceHolderID="<TemplatePlaceHolderID>" ↵
runat="server">
    .
    .
                <YOUR CONTENT HERE>
    .
    .
</asp:Content>
```

Creating a New Template

During the process of creating a new Web application, you are requested to select a template. By default, these templates are ready to use Web site designs. You can create a template and add it to the list of available templates.

Design Tips

- Before making changes in the css files, it is highly recommended to backup these files.

- Changing colors in the css files can effect the entire look and feel of the application, therefore it is recommended to test the changes on a few different screens.

- Rather than changing existing classes to meet the template requirements, try adding new classes of your own.

- Instant pages rely on absolute positioning defined in the styles_Instant.css. Do not change position style attributes in the css files.

- Pages generated with absolute positioning: changing position attributes in the style_generated.css file may cause them not to display properly (this can be overcome in design time).

- Remember that changes made to one css file will usually need to be implemented to all three css files, if both generated pages and instant pages are to have the same look and feel.

▶ **To create a new template:**

1   Create a new Web application using one of the built-in templates.

2   Design the template.jsp/template.master file as desired.

Ensure that the template includes:

JSP: <gx:placeholder id="PagePlaceHolder"/> tag.

.NET: <asp:ContentPlaceHolder ID="GXPagePlaceHolder" runat="server"/> tag.

This tag's position on the screen can vary and can be placed wherever you wish. Removing this tag means that screen content for generated and instant pages will not be displayed.

3   Package the template and place it in the installation directory, so that it will be displayed when using the Create new Web Application wizard:

1.   Create a new directory in the Templates directory in the ApplinX installation folder. The name of the directory will be the name of the template displayed in the New Web Application wizard.

2.   Within this directory create a common directory and a java/dotnet (according to the template you created) directory. Within the common directory, create a css and images directory:
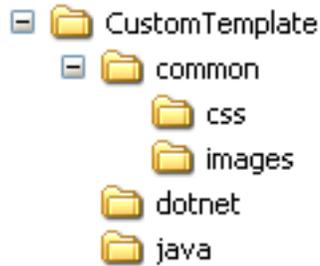
3. In the css folder place all the css files that you modified/created.

4. In the images folder place all the image files that you modified/created in the template file or in the css files.

5. In the common folder place the image that the wizard will display as a preview of this template. The image size should be 630x400pixels and the name of the image file must be thumbnail.jpg.

6. Place the template file in the relevant java/dotnet folder.

Activating an Application Map from a Menu

Navigation menus, are very common in Web applications. They provide a direct, quick navigation to any page in the application. In ApplinX Web applications, navigation menus can be used to directly access any available screen in the host application, thus eliminating the need of using the original menu screens in the host while fully preserving their functionality. This is achieved using the application map. Each menu option will be an accessible host screen, and clicking it will invoke the ApplinX application's map with the screen's name as its parameter.

Refer to the Composite demo, template.jsp/template.master.cs in which the proposals/customers links set a hidden field with the desired screen name and invoke a server side function (`beforePaneMenuBtn_Click`) that performs the navigation to the desired screen (refer to /GXBasic-Context.jsp/template.master.cs).

Using ApplinX Repository Folders' Structure to Organize Web Pages

ApplinX Web application's Web pages can be hierarchically organized according to the folder structure of the screens they represent. For example, if the screen *MainMenu* resides in the folder *general\menu_screens* in the ApplinX repository, it can reside under the directory *\root folder\general\menu_screens* in the Web application.

▶ **To use ApplinX repository folder structure to organize Web pages:**

1   Refer to Configuring your Framework and access the Framework Configuration Editor.

2   Ensure that the **Use Folders** check box in the **General** node is selected. The field **Virtual directory** is displayed.

3   Enter the name of the virtual directory of the Web application. For example: CompositeDemo.

4   To generate a Web page into a specific folder:

   1. Create a directory structure under the root folder of the Web application that corresponds to the folder structure of the ApplinX repository.

   2. You can either select the directory within the generation wizard (JSP only), or you can move the generated pages to the relevant directory (JSP and .Net).

Controlling the Connection Properties from the Code

ApplinX allows passing various connection parameters from the code in order to control the configuration of the application.

For example: Controlling from the code if the application will work against a trace file.

Refer to the Composite demo (with/without a trace file).

.NET:

```
gx_appConfig.SessionConfig.addVariable(GXBaseObjectConstants.GX_VAR_REPLAY_FILE,null);

   // to cancel the replay file configured
```

JSP:

```
getGXAppConfig().getSessionConfig().addVariable(GXBaseObjectConstants.GX_VAR_REPLAY_FILE,null);

   // to cancel the replay file configured
```

For a list of possible connection variables, see GXBaseObjectConstant class.

For code sample, see Composite Demo Application: "Login" path procedure - ApplinX repository Login.aspx / Login.aspx.cs - .NET Login.jsp / WEB-INF/classes/contexts/Login.java – JSP

Handling Flickering of Screens

It is necessary to use the Flickering of Host Sessions feature when one of the following happens:

- In the browser, a blank screen (empty screen) is displayed when navigating between two host screens.

- In the browser you are required to submit the [ENTER] key (or any other key) twice in order to navigate to the next host screen.

The initial need for Flicker arises when specific host screens are received 'split' between several buffers of data. Thus ApplinX Server needs to be informed to wait an additional amount of time for the complete screen to arrive. This additional amount of time is defined (in milliseconds) in the Flicker parameter in the Application Configuration dialog box.

The flicker setting applies to the entire ApplinX application, meaning that if the flicker is set to 500ms, after each host transaction the flicker time will be added to the communication time. In other words, the entire application will be 'slowed down' by the flicker time. Therefore this value should only be set for the entire application according to the guidelines detailed in Handling Flickering of Screens. Also refer to Waiting for Screens using Wait Conditions and to the **Perform background check for host screen changes** parameter in the Configuration Editor for further details.

Waiting for Screens, using Wait Conditions

Wait conditions are used to handle specific 'problematic' screens that require waiting an additional amount of time for the complete screen to arrive. Refer to Handling Flickering of Screens for further details regarding using the wait conditions.

To add a wait condition that waits for a specific screen in the application:

JSP

In *GXBasicContext.java*, in the method `gx_postSendKey`, add the following code:

```
GXGetScreenRequest sr = new GXGetScreenRequest();
//An example of adding a wait condition in a page that waits for specific screens ↵
according to the key sent.
if(gx_context.getGXForm().get_HostKeys().equalsIgnoreCase("[PF3]")
&& ↵
gx_context.getGXSession().getScreen().getName().equalsIgnoreCase("CustomerDetails"))
{
    sr.addWaitCondition(new GXWaitForScreen("MainMenu",50000,0));
    gx_context.getGXSession().getScreen(sr);
}
```

.NET

In *GXBasicWebForm.cs,* in the method `user_postSendKeys`, add the following code:

```
GXGetScreenRequest sr = new GXGetScreenRequest();
//An example of adding a wait condition in a page that waits for specific screens ↵
according to the key sent.
if(gx_form.HostKeys.Equals("[PF3]")
&&  gx_session.getScreen().getName()=="CustomerDetails")
{
    sr.addWaitCondition(new GXWaitForScreen("MainMenu",50000,0));
    gx_session.getScreen(sr);
}
```

To apply a flicker setting to a certain action:

JSP

In *GXBasicContext.java,* in the method `gx_postSendKey`, add the following code:

```
GXGetScreenRequest sr = new GXGetScreenRequest();
if(gx_context.getGXForm().get_HostKeys().equalsIgnoreCase("[enter]")
&& ↵
gx_context.getGXSession().getScreen().getName().equalsIgnoreCase("CustomerDetails"))
{
    long lTimeout = 50000;//millisecond
    long lFlicker = 750;//millisecond
    sr.addWaitCondition(new GXWaitHostQuiet(lTimeout,lFlicker));
    gx_context.getGXSession().getScreen(sr);
}
```

.NET

In *GXBasicWebForm.cs,* in the method `user_postSendKeys`, add the following code:

```
GXGetScreenRequest sr = new GXGetScreenRequest();
if(gx_form.HostKeys.Equals("[enter]")
&&  gx_session.getScreen().getName()=="CustomerDetails1")
{
    long lTimeout = 50000;//millisecond
    long lFlicker = 750;//millisecond
    sr.addWaitCondition(new GXWaitHostQuiet(lTimeout,lFlicker));
    gx_session.getScreen(sr);
}
```

Customizing the Host Keys

It is possible to display the host keys for the current host screen (analyzed by ApplinX Server according to the defined host keys patterns) as Web buttons, hyperlinks or in other advanced formats (images, etc.). In addition, it is possible to fully customize the host keys using the GXIHostKeysTagUserExit interface (adding buttons, removing links, changing captions and more). To make complex changes to the control, such as adding a row/column, implement the interface GXIHostKeysTagUserExit as necessary.

Refer to Host Keys for configuring a host key pattern.

▶ **To display and customize the host keys:**

1   By default, after defining a host key pattern, the analyzed keys are displayed as hyperlinks in the Web application.

2   To display the keys as buttons, refer to Configuring your Framework and access the Framework Configuration Editor. Select Buttons in the Host keys field in the Instant node.

3   Customization of the host keys' appearance (colors, fonts etc.) is done in the style sheets (*css\styles_instant.css* and *css\styles_generated.css*), in the gx_hky class.

4   For advanced customization, it is possible to use the Host Keys' control. By using the template option of the control, it is possible to write a custom template for displaying the host keys. The following code is usually placed in one of the template jsp/aspx files and displays a different image for each host key; the image name is determined according to its caption. In addition, the keys are positioned vertically:

JSP

```
<gx:hostKeys vertical="true" border="0" keyType="template">
        <a href="#" onclick="gx_SubmitKey('$(ACTION)')" ↵
title="$(ACTION)-$(CAPTION)">
        <img src="images/keybuttons/btn$(CAPTION).gif" border="0"/>
        </a>
</gx:hostKeys>
```

.NET

```
<gx:GXHostKeysControl runat="server" vertical="true" border="0" ↵
KeyType="Template">
        <a href="#" onclick="gx_SubmitKey('$(ACTION)')" ↵
title="$(ACTION)-$(CAPTION)">
        <img src="images/keybuttons/btn$(CAPTION).gif" border="0"/>
        </a>
</gx:GXHostKeysControl>
```

In addition, add the following import statement to the file containing the control: `<%@ Register TagPrefix="gx" Namespace="com.sabratec.dotnet.framework.web.controls" Assembly="GXDotnet" %>`

▶ **To fully customize the host keys control:**

1 **JSP**

. In the *GXBasicContent.java* file, the `gx_onInit` method, add the following line:

```
getGXAppConfig().setHostKeysTagUserExit(new ↵
transforms.UserHostKeysTagTransform());
```

.NET

: In the *GXBasicWebForm* file, the `OnInit` method, add the following line:

```
gx_appConfig.setHostKeysTagUserExit(new transforms.UserHostKeysTagTransform());
```

2 In your Web application, transforms directory, open the *userHostKeysTagTransform* file (when upgrading your application, copy the file from the relevant framework type new application) and implement the `onHostKeysComplete` interface.

For example JSP:

```
public void onHostKeysComplete(GXITableTag hostKeysTableTag, GXIScreen screen) {

        super.onHostKeysComplete(hostKeysTableTag, screen);

        /*
        // add link
        addLink("Host help","gx_SubmitKey('[pf1]')");

        //get all tags
        GXITag[] tags = getHostKeysTags();

        for (int index=0; index<tags.length; index++){
              //get tag
```

```
            GXITag tag = tags[index];

        if (tag instanceof GXILinkTag){ // or GXIButtonTag or GXHtmlString ↵
(for type="template")
                GXILinkTag link = (GXILinkTag) tag;

                // change tag text
                if (link.getText().indexOf("Prev") >= 0){
                        link.setText("Previous");
                }

                //remove tag
                if (link.getText().indexOf("Pagedn") >= 0){
                        removeTag(link);
                }
            }
        }
        */

    }
```

For example .NET

```
public override void onHostKeysComplete(GXITableTag hostKeysTableTag, GXIScreen ↵
screen)
    {
        /*
        base.onHostKeysComplete(hostKeysTableTag, screen);

        //add link
        addLink("Host help", "gx_SubmitKey('[pf1]')");

        //get all tags
        GXITag[] tags = getHostKeysTags();

        //loop on controls in hostKeys array
        foreach (GXITag tag in tags)
        {
            if (tag is GXILinkTag)  // or GXIButtonTag or GXHtmlString (for ↵
type="template")
            {
                //get link
                GXILinkTag link = (GXILinkTag)tag;

                // change control text
                if (link.getText().IndexOf("Prev") >= 0)
                {
                    link.setText("Previous");
                }

                //remove control
                if (link.getText().IndexOf("Pagedn") >= 0)
```

```
            {
                removeTag(link);
            }
        }
    }
    */
}
```

For complete description of the Host Keys control properties, see Host Keys Component.

Sample code can be found in the Instant Demo application. *Template.jsp , template.master (.NET)* – contains an example for using the Host Keys control.

Activating the Screen Locker

The ApplinX framework contains a built-in feature of a screen locker. The purpose of a screen locker is to indicate to the user by means of a message, that the application is processing his request, and blocks him from interfering with the current process by repressing a button/link or keyboard PF/ENTER.

▶ **To activate the screen locker:**

1 Refer to **Configuring your Web Application** and access the Framework Configuration Editor. In the **General** node, select **Use screen locker**.

2 Use the width/height percentages in the file *template/screenLocker.htm* to control the location of the message.

3 Replace the text "Please wait" with an alternative text/image as required.

> **Note:** When adding a link in the page that performs a JavaScript action it is highly recom-
> mended to use `onclick` event and not the href attribute. Using the href attribute will cause
> the screen locker to be activated.

It is also possible to control the screen locker on the Web page level (refer to **Handling the Screen Locker on the Page Level**).

Natural UNIX: Integrating a Login Page in the Web Application

When the Natural UNIX host requires separate connection for each individual user together with support of password changing/expiring, the ApplinX framework provides a suitable Login Page to be used in your Web application. The Login page should be displayed before attempting to connect to the host. In this page the user is required to enter the user name and password. Only after the user name and password are authenticated by the host, the connection to the host is made. The look and feel of the Login page can be edited and changed to suit your Web application. The login page logic is implemented within the ApplinX framework.

> **Note:** The hostLogin page is provided with your Web application but can also be created using the Base Object. Refer to the Start>Programs>Software AG ApplinX>Documentation>ApplinX Development API for further information.

▶ **To use the Login page:**

■ Edit the `index.jsp/index.aspx` file and instead of `gxfirstpage.jsp/gxfirstpage.aspx`, enter `hostLogin.jsp/hostLogin.aspx`, or refer your end users to this page.

Implementing & Controlling JavaScript Events using the gx_event Object

The ApplinX JavaScript engine provides a cross browser event object that enables creating dynamic web pages. The gx_event object is created within the Javascript engine. Whenever an event is triggered it is passed to the user Exit functions (/js/userExits.js) and from there to the page-level function (if one exists).

gx_event enables handling browser events such as OnKeyDown. The following example will cancel the `OnKeyDown` event whenever the [Enter] key is pressed on a certain text area ("myTextArea"), prevent the page from being submitted and manually add a newline character to the text area value:

Assume your JSP/ASPX page has the following input:

```
<textarea row="5" id="myTextArea" ></textarea>
```

Add the following `pageOnKeyDown` function to your generated page

```
function pageOnKeyDown(gx_event){
 ...
 var win = gx_event.window;
 if (gx_event.keyCode==13 && gx_event.element.id=="myTextArea"){
  gx_event.cancel();
  GXBrowserUtil.getElement("myTextArea").value += "\r\n"
 }
...
}
```

Refer to the API:

- `GXEvent_Object`

- `GXEvent.keyCode`

- `GXEvent.additionalKey`

- `GXEvent.element`

Retrieving Data from Fields Outside the Modal Window Currently Displayed

Relevant data, such as host messages or errors, may sometimes appear in the area outside the currently displayed modal window frame. To retrieve these messages/errors it is necessary to retrieve the entire screen content rather than just the modal window content. This can be achieved by writing the following method:

Java

```
private String getMessgeOutsideWindow(int MessageRow, int MessageCol){
     try {
             // Create a Screen Request the would ignore modal window definition
             GXGetScreenRequest gsr = new GXGetScreenRequest();
             ↵
gsr.addVariable(GXBaseObjectConstants.GX_VAR_HOST_WINDOW_ENABLED,"false");

             // Get the current screen from ApplinX server using the Screen request
             GXIScreen screen = getGXSession().getScreen(gsr);

             // Return Field content according to the specified position
             GXPosition pos = new GXPosition(MessageRow, MessageCol);
             return screen.getFields().getFieldByPosition(pos).getContent();

     } catch (GXGeneralException e) {
             return null;
     }
}
```

.NET

```
private string getMessgeOutsideWindow(int MessageRow, int MessageCol)
{

     // Create a Screen Request the would ignore modal window definition
     GXGetScreenRequest gsr = new GXGetScreenRequest();
     gsr.addVariable( GXBaseObjectConstants.GX_VAR_HOST_WINDOW_ENABLED,  "false");

     // Get the current screen from ApplinX server using the Screen request
     GXIScreen screen = gx_session.getScreen(gsr);

     // Return Field content according to the specified position
     GXPosition pos = new GXPosition(MessageRow, MessageCol);
     return screen.getFields().getFieldByPosition(pos).getContent();
}
```

> **Note:** This method can be added to `GXBasicContext` or to a generated page's code class.

ApplinX Server Load Balancing

When you have several ApplinX servers and want new sessions to always be created on the least busiest server, determine this using the GXLoadBalancer class.

▶

1   From within the relevant application in the new JSP directory, open *WEB-INF\classes\con-texts\GXBasicContext*.

2   Add the necessary statements:

   JSP

   ```
   import com.sabratec.ci.GXLoadBalancer
   ```

   .NET

   ```
   using com.sabratec.dotnet.ci;
   ```

3   Within the file, locate the `gx_initSessionConfig` method and place the following code:

```
/// ApplinX Server Load Balancing   /////////////////////////////////
    GXWebAppConfig gx_appConfig = getGXAppConfig();

    GXLoadBalancer lb = new GXLoadBalancer();
    // Add server URLs. Example:
    // lb.addServerURL("applinx://localhost:2323");
    lb.addServerURL("applinx://<ApplinXServer1>:<portNumber>");
    lb.addServerURL("applinx://<ApplinXServer2>:<portNumber>");
    lb.addServerURL("applinx://<ApplinXServer3>:<portNumber>");
    lb.addServerURL("applinx://<ApplinXServer4>:<portNumber>");

// Replace ApplinXServerX with the server name
// Replace portNumber with the appropriate ApplinX server port number.

    String availableServer = lb.getAvailableServer();
    gx_appConfig.getSessionConfig().setServerURL(availableServer);
```

Retrieving a Unique Device Name per User Name / IP Address from a Database and Applying it to the Current User Session

Some hosts require that every user connection must supply a device name in order to establish a connection. These device names may be unique per user or per IP address. In such cases, you will need to retrieve the device name from the place where your organization stores it (database, LDAP, XML etc.).

The following example will demonstrate how to retrieve the device name from the database (MDB) using the user's IP address and how to set the retrieved value to the ApplinX session, prior to connecting to the host.

JSP

Add the following method to *GXBasicContext.java*:

```
// This method retrieves the device name from an MDB database.
// You can change this code to match your type of database.
public String getDevice(String userIP) {
  String Result = "";
  try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // set this to a MS Access DB you have on your machine
    String filename = "C:\\users.mdb";
    String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";

    database+= filename.trim() + ";DriverID=22;READONLY=true}";    // add on to ↵
the end
    // now we can get the connection from the DriverManager
```

```
        System.out.println(database);
        Connection con = DriverManager.getConnection(database , "", "");
        Statement stmt = con.createStatement();
        ResultSet rs;

        // Retrive Device name according to user's IP address
        rs = stmt.executeQuery("Select Device from userDevices where userIPAddr='" + ↵
userIP + "'");

        while ( rs.next() ) {
            Result = rs.getString("Device");
        }
        con.close();
    } catch (Exception e) {
            System.err.println(e.getMessage());
    }
    return Result;
}
```

In the this file, edit the `gx_initSessionConfig` method as follows:

```
public void gx_initSessionConfig(){
    // for device name
    GXWebAppConfig gx_appConfig = getGXAppConfig();
    String ipAddr = getRequest().getRemoteAddr();
    gx_appConfig.getSessionConfig().addVariable(
            GXBaseObjectConstants.GX_VAR_DEVICE_NAME, getDevice(ipAddr));
}
```

.NET

Add the following method to *GXBasicWebForm.cs*:

```
private string getDevice(string userIP)
{
    OdbcConnection con = new OdbcConnection();
    con.ConnectionString = "Driver={Microsoft Access Driver ↵
(*.mdb)};Dbq=c:\\users.mdb;Uid=sa;Pwd=sa;";
    con.Open();
    string result="";
    try
    {
        // Retrive Device name according to user's IP address
        OdbcCommand cmd = new OdbcCommand("Select Device from userDevices where IP='" ↵
+ userIP + "'", con); // creating query command
        OdbcDataReader reader = cmd.ExecuteReader(); // executes query
        while (reader.Read()) // if can read row from database
        {
            result = reader["Device"].ToString();
```

```
      }
   }
   catch (Exception ex)
   {
      // shows exception message in console if any errors occured
      Console.WriteLine("Ex: " + ex.Message);
   }
   finally
   {
      con.Close();  // finally closes connection
   }
   return result;
}
```

In the this file, edit the `gx_initSessionConfig` method as follows:

```
public void gx_initSessionConfig(){
   // for device name
   String ipAddr = Request.UserHostName;
   gx_appConfig.getSessionConfig().addVariable(
        GXBaseObjectConstants.GX_VAR_DEVICE_NAME, getDevice(ipAddr));
}
```

Retrieving the Host Printer Device Name from a Database and Setting the ApplinX Printlet to Work with that Device

Connecting to a host printer queue requires users to supply a device name. The device provides the host information as to which printer queue to associate each user. Usually, each user or IP address has a pre-defined device name. This data can be stored in a database and retrieved before creating the printer session in the ApplinX web application.

The following example will demonstrate how to retrieve the device name from the database and set it to the appropriate printlet parameter.

JSP

Replace the `getDevice` method in *RunPrintlet.java*:

```
// This method retrieves the device name from an MDB database.
// You can change this code to match your type of database.
public String getDevice(String userIP) {
   String Result = "";
   try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // set this to a MS Access DB you have on your machine
      String filename = "C:\\users.mdb";
      String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=";

      database+= filename.trim() + ";DriverID=22;READONLY=true}";    // add on to ↵
the end
      // now we can get the connection from the DriverManager
      System.out.println(database);
      Connection con = DriverManager.getConnection(database , "", "");
      // String url = "jdbc:odbc:users.mdb";
      // Connection conn = DriverManager.getConnection(url,"","");
      Statement stmt = con.createStatement();
      ResultSet rs;

      rs = stmt.executeQuery("Select Device from userDevices where userIPAddr='" + ↵
userIP + "'");

      while ( rs.next() ) {
         Result = rs.getString("Device");
         System.err.println(Result);
      }
      con.close();
   } catch (Exception e) {
         System.err.println(e.getMessage());
   }
   return Result;
}
```

In *run_Printlet.jsp* set the device name parameter as follows:

```
<param name="device_name" value="<%=new ↵
RunPrintlet().getDevice(request.getRemoteAddr()) %>">
```

.NET

Replace the getDevice method in *run_printlet.aspx.cs*:

```
private string getDevice(string userIP)
{
   OdbcConnection con = new OdbcConnection();
   con.ConnectionString = "Driver={Microsoft Access Driver ↵
(*.mdb)};Dbq=c:\\users.mdb;Uid=sa;Pwd=sa;";
   con.Open();
   string result="";
   try
   {
      // Retrive Device name according to user's IP address
      OdbcCommand cmd = new OdbcCommand("Select Device from userDevices where IP='" ↵
+ userIP + "'", con); // creating query command
      OdbcDataReader reader = cmd.ExecuteReader(); // executes query
      while (reader.Read()) // if can read row from database
      {
         result = reader["Device"].ToString();
      }
   }
   catch (Exception ex)
   {
      // shows exception message in console if any errors occured
      Console.WriteLine("Ex: " + ex.Message);
   }
   finally
   {
      con.Close();  // finally closes connection
   }
   return result;
}
```

In *run_printlet.apx.cs* edit the `Page_Load` method and add the following assignment:

```
// Retrieve Device name By User's IP
GXPrinterControl1.deviceName = getDevice(Request.UserHostName);
```

Notes:

1. You can implement the same methodology for of the printlet parameters.

2. If ApplinX server and the web application's server do not reside on the same machine, refer to Configuring the Printlet Redirector

Customize ApplinX Framework Session Error Handling

The default behavior of the ApplinX framework when it encounters an ApplinX session error is to redirect the user to the *error.jsp* page. This page will specify the error code and will also provide a description of the error. For example: If, for some reason, the ApplinX session has been disconnected, pressing a PF key in the web application will result in the following error:

However, in some cases you may want to capture these errors and perform a different action. The following example will demonstrate how the default ApplinX method can be overridden with new functionality.

For example, upon receiving a 5001 error, you may want the web application to automatically try to re-establish a connection to the host.

JSP

To do this, add the following code to the `GXBasicContext` class:

```
// Override the gx_handleSessionError with additional functionality
public void gx_handleSessionError(GXGeneralException ex){
 try{
  // Is the error code 5001?
  if (ex.getErrorCode()== 5001){
   // Try to re-establish a connection to the host
   getResponse().sendRedirect("gxfirstpage.jsp");
  }
  else{ // Otherwise, perform the default behavior
   super.gx_handleSessionError(ex);
```

```
  }
 }
catch (IOException e){
 // TODO : Handle Failed page redirection
 }
}
```

.NET

To do this, add the following code to the `GXBasicContext` class:

```
public override void ↵
gx_handleSessionError(com.sabratec.applinx.baseobject.GXGeneralException err)
{
   if (err.getErrorCode() == 5001)
   {
      // Try to re-establish a connection to the host
      Response.Redirect("gxfirstpage.aspx");
   }
   else
   { // Otherwise, perform the default behavior
      base.gx_handleSessionError(err);
   }
}
```

Customize the Web Application's Error Page

One may want to customize the Web application's error page for any number of reasons. These reasons may include providing the error page in non-English languages, or to change the look and feel of the page or add additional messages.

The look and feel of the page can easily be customized by changing the layout of the error.jsp page. Changing the page's dynamic content can be achieved by editing the *error.java* file.

The *error.java/Error.apsx.cs/vb* file contains an example for translating a specific error message to another language, according to the user's locale setting. In order to activate this functionality un-comment the CustomizeErrorHandling method and un-comment the call to CustomizeErrorHandling in `gx_onLoad` in the *error.java* file (JSP) or `Page_Load` in the error.aspx.cs file (.Net).

12 Transferring Natural Data to/from the Host

The host sometimes requires uploading/downloading files. Part of the process can be performed automatically by defining to work with an ActiveX component or when setting the download to Automatic mode (relevant when working with Mozilla browsers, as it is not possible to work with an ActiveX component with this browser). These settings can be configured in the **Framework Configuration Editor**, in the **Natural upload/download** node.

> **Note:** It is required to define Natural Transfer Support in the Host Properties, Options tab.

Transferring Data using ActiveX

> **Notes:**

1. ActiveX will work only if you have Microsoft .NET Framework version 3.5/4.0 installed on your machine.

2. The site using the ActiveX needs to be defined as a "trusted site". Under **Tools** > **Internet Options** > **Security**, click **Custom level** and under **ActiveX controls and plug-ins** > **Allow Scriptlets** check **Enable**.

- Downloading Data
- Uploading Data
- Configuring the Security Tab for Internet Explorer

Downloading Data

When the host refers to a specific file name, the file will be downloaded automatically (silently).

When the host does not specify the file name and location, a Windows dialog box will indicate where the file should be saved .

It is possible to determine that the file will automatically overwrite an existing file (without requiring the user to confirm this action). This setting can be configured in the Framework Configuration Editor, in the Natural upload/download node.

Uploading Data

When the host provides a name, the file is uploaded automatically (silently).

When the host does not provide a name the Windows File selection dialog box is displayed so that you can select the file to be uploaded.

Configuring the Security Tab for Internet Explorer

To install the the ActiveX control on Internet Explorer you will need to configure the security tab.

▶ **To adjust ActiveX settings for Internet Explorer**

1   Open Internet Explorer.

2   Click the **Tools** menu, and then click **Internet Options**.

3   On the **Security** tab, choose **Trusted site** and add the site to the trusted site list.

4   Click on **Custom** level and scroll down the **Security Settings** list until you see ActiveX controls and plug-ins.

5   For Automatic prompting for ActiveX controls, click **Enable**.

6   Scroll down to **Download signed ActiveX controls** and click **Enable** or **Prompt**.

7   Scroll down to **Run ActiveX controls and plug-ins** and click **Enable** or **Prompt**.

8   Scroll down to **Script ActiveX controls marked safe for scripting** and click **Enable** or **Prompt**.

9   Scroll down to **Automatic prompting for ActiveX controls** and click **Enable**, Click **OK**, and then click **OK** again.

After you have completed these steps, log on to the application to run it again.

Transferring Data without using ActiveX

- Downloading Data
- Uploading Data

Downloading Data

When the host refers to a specific file name, the download will start automatically, and when the downloading process has been completed, an Open/Save browser dialog box is displayed.

When the host does not specify the file name and location, the ApplinX Download dialog box is displayed where you specify the file name and extension (It is possible to determine that this screen will not be displayed and that the download will start automatically by selecting **Automatic download** in the **Natural upload/download** node of the **Framework Configuration Editor**). Once the downloading process is completed, an Open/Save dialog box is displayed.

Uploading Data

The ApplinX Upload dialog box is displayed. Browse and select a file to upload and then click **Upload**. When the host provides a name, a message is displayed with the requested file name

Uploading Large Files (more than 4MB)

JSP

When uploading files using JSP, you may need to allocate additional memory to your Web server. This is a Web Server limitation and not ApplinX.

.NET

When uploading files using .NET add the following line to your *web.config file* as a child of `<system.web>`:

```
<httpRuntime executionTimeout="90" maxRequestLength="4096"
useFullyQualifiedRedirectUrl="false" minFreeThreads="8"
minLocalRequestFreeThreads="4" appRequestQueueLimit="100"/>
```

Note that the default value of the maxRequestLength attribute is 4096 (4MB). Change the value to suit your needs.

Activating a Natural Command

When using ActiveX in the framework for Natural Data Transfer (see Configuration Editor), Natural commands of type OS/DOS are executed on the browser machine automatically by the ApplinX framework. When not using ActiveX the command is not handled

13   Instant Pages Customization

Using a Proportional (Non-Fixed) Font in Web Pages

By default, ApplinX uses a monospaced, fixed font (Courier New) for displaying instant HTML pages. However, it is possible to use a proportional (non-fixed) font, to achieve a more modern, Web-like look & feel. Since the rendering of the HTML screen is based on absolute positioning, using a proportional font usually does not damage the correct alignment of the host fields (although some data may appear misaligned).

▶ **To use a proportional font in Web pages (JSP & .NET):**

■ **For Instant HTML Pages**

In the file *\css\styles_instant.css*, change the following classes to use a proportional font of your choice, for example:

```
#gx_screenArea
{
        position:relative;
        font-family: "Verdana";
        height:0px; /* make scrolling when needed*/
}
#gx_screenArea input,
#gx_screenArea select {
        font-family: "Verdana";
        position:absolute
}
```

Or:

For Generated Pages

In the file *\css\styles_generated.css*, in a similar manner as for instant HTML pages, change all the classes that use the font "courier new" to a proportional font.

Controlling Instant Display Properties

It is possible to set various display properties of instant HTML pages, such as colors, fonts, etc. This is usually done in order to apply a Web look & feel to the ApplinX Web application.

Most of these settings are done in the style sheet file. Specific settings are set in other files. Refer to the relevant tasks for more details.

▶ **To control instant display properties (JSP and .NET):**

1   The *css\styles_instant.css* file contains all CSS classes used for rendering instant HTML pages. Change them as necessary to affect the colors and fonts used for the pages.

2   To find out which CSS class is used for a specific element or field in the instant HTML page, it is possible to use the Browser view source option to view the generated HTML source, and then look for the element and extract its CSS classes.

> **Note:** This style sheet applies only to instant HTML pages. For modifying the styles used for generated web pages (JSP, ASPX), use the *\css\styles_generated.css* file.

Code Transformations

A Code Transformation (or Code Transform) is a code class (written in Java, C# or VB.NET for JSP or .NET frameworks respectively), that is executed on all host screens or on a specific group of screens (i.e. an ApplinX Screen Group) and modifies the rendered instant HTML page for the current host screen it's executed on.

Refer to

- **Creating a New Code Transformation**
- **Applying a New Transformation to all Screens**
- **Applying a New Transformation to a Screen Group**
- **Manipulating Individual Host Fields**

- **Positioning Specific Fields**
- **Formatting Specific Fields**
- **Replacing a Field's Text**
- **Replacing a Field with a Web Element, Adding a Web Element**
- **Manipulating Host Characters**
- Manipulating Host Keys

Creating a New Code Transformation

In most cases, transformations are defined using the Transformation Wizard . However, when due to limited flexibility, it is not be possible to define certain required transformations using the wizard, complement the wizard-defined Transformation entities with code-defined Transformation classes.

Creating a new Instant transformation is carried out by using the provided template transformations (`UserTagTransform1`, `UserCompletionTransform1`, `UserHostKeysTagTransform`).

`UserTagTransform1` contains events for each individual rendering event and should be used for new transformations that manipulate specific tags (input fields, output fields, GUI elements, etc.) – for example, adding images or repositioning fields.

`UserCompletionTransform1` is called upon completion of the entire rendering process and should be used for new transformations that manipulate the entire rendered HTML – for example, a transformation that converts a host menu screen to a list of hyperlinks.

▶ **To create a new transformation (JSP):**

1   Copy one of the template transformations from *\web-inf\classes\transforms* and rename the file and the class:

```
public class SampleTransform extends GXTagListener{
```

2   Add code in the relevant stub methods.

3   To apply the transformation, see the tasks **Applying a New Transformation to all Screens** and **Applying a New Transformation to a Screen Group**.

▶ **To create a new transformation (.NET):**

1   Copy one of the template transformations from the *transforms* directory and rename the file and the class:

```
public class SampleTransform: GXTagListener
```

2   Add code in the relevant stub methods.

3   To apply the transformation, see the **Applying a New Transformation to all Screens** and **Applying a New Transformation to a Screen Group**.

Applying a New Transformation to all Screens

This task applies an existing transformation. To create a new transformation, see the task **Creating a New Transformation**.

▶ **To apply a new transformation to all host screens (JSP):**

■   In the file *web-inf\classes\contexts\GXInstantLogicContext.java*, uncomment the code in the function `registerInstantTransforms` and register the new transformation class:

```
public void registerInstantTransforms()    {
          GXRenderConfig instantConfig = getGXAppConfig().getInstantConfig();
            instantConfig.addTagListener(new UserTagTransform1());
  // add here more transform registrations
}
```

`UserTagTransform1` being the name of the new transformation class.

▶ **To apply a new transformation to all host screens (.NET):**

■   In the file *GXInstantLogicWebForm.cs*, uncomment the code in the function `registerInstantTransforms` and register the new transformation class:

```
public virtual void registerInstantTransforms() {
        gx_appConfig.InstantConfig.addTagListener(new UserTagTransform1());
        // add here more transform registrations
}
```

`UserTagTransform1` being the name of the new transformation class.

Applying a New Transformation to a Screen Group

This task applies an existing transformation. To create a new transformation, refer to **Creating a New Transformation**.

It is possible to apply a transformation only for a specific Screen Group (or several Screen Groups). For example, a transformation that handles the command line field should be applied to the screen group of all screens containing this field.

▶ **To apply a new transformation to a screen group (JSP):**

1   If the screen group has its own generated Web page, add the registration function to the file *web-inf\classes\contexts\screenGroupName.java*, to the function `registerInstantTransforms`:

```
public void registerInstantTransforms() {
super.registerInstantTransforms();
GXRenderConfig instantConfig = getGXAppConfig().getInstantConfig();
        instantConfig.addTagListener(new MyTagTransform());
        instantConfig.addCompletionListener(new MyCompletionTransform());
}
```

Where `MyTagTransform` is the name of the new transformation class and `MyCompletionTransform` is the name of a new completion transformation class.

2   If the screen group does not have its own generated Web page, register it in *web-inf\classes\contexts\GXInstantLogicContext.java,* and add code that will register it only for the appropriate screen group:

```
public void registerInstantTransforms() {
GXRenderConfig instantConfig = getGXAppConfig().getInstantConfig();
try {
if (getGXSession().getScreen().isMemberOf("MyScreenGroup")) {
    instantConfig.addTagListener(new MyTagTransform());
    instantConfig.addCompletionListener(new MyCompletionTransform());
}
} catch (GXGeneralException e) {}
}
```

Where `MyScreenGroup` is the name of the screen group, `MyTagTransform` is the name of a new tag transformation class and `MyCompletionTransform` is the name of a new completion transformation class.

3   It is also possible to add code in the transformation class itself that will activate it only for the appropriate screen group(s).

▶ **To apply a new transformation to a screen group (.NET):**

1   If the screen group has its own generated Web page, add the registration function to the file *screenGroupName.aspx.cs*, to the function `registerInstantTransforms`:

```
public override void registerInstantTransforms() {
base.registerInstantTransforms();
gx_appConfig.InstantConfig.addTagListener(new MyTagTransform ());
gx_appConfig.InstantConfig.addCompletionListener(new MyCompletionTransform());
}
```

Where `MyTagTransform` is the name of the new transformation class and `MyCompletionTransform` is the name of a new completion transformation class.

2   If the screen group does not have its own generated web page, register it in *web- GXInstantLogicWebForm.cs*, and add some code that will register it only for the appropriate screen group:

```
public virtual void registerInstantTransforms() {
if (gx_session.getScreen().isMemberOf("MyScreenGroup")) {
        gx_appConfig.InstantConfig.addTagListener(new MyTagTransform ());
gx_appConfig.InstantConfig.addCompletionListener(new MyCompletionTransform());
}
}
```

Where `MyScreenGroup` is the name of the screen group, `MyTagTransform` is the name of a new tag transformation class and `MyCompletionTransform` is the name of a new completion transformation class.

3   It is also possible to add code in the transformation class itself that will activate it only for the appropriate screen group(s).

Manipulating Individual Host Fields

The following tasks handle manipulation of specific host fields. The common methodology for such manipulation is identifying the screen groups including these fields, mapping the relevant fields as application fields and writing custom transformations for handling the mapped fields. See also Instant Web Application Development Methodology.

📄   **Note:** It is possible to carry out basic manipulation of fields using the Transformation Wizard.

Positioning Specific Fields

It is possible to reposition a specific field in a different position in the Web page (instead of in its original host position). It is also possible to display a field in one of the template sections.

> **Note:** It is recommended to map the fields as application fields. Refer to the Instant Web Application Development Methodology section for general instructions on mapping fields to screen groups.

> **Note:** It is possible to carry out basic manipulation of fields using the Transformation Wizard, Text to Text transformation or Input Field to Text Field transformation (detailed in the Transformations).

▶ **To position a specific field:**

1   To reposition a field, create a new tag transformation and register it to the relevant screen group(s) or to all screens. Refer to Creating a New Transformation, Applying a New Transformation to all Screens and Applying a New Transformation to a Screen Group.

2   In the transformation class, add code that will reposition the field in the appropriate method. It will usually be `onNewLabel` (for output fields) or `onNewTextField` (for input fields):

JSP

```
public void onNewLabel(GXRenderEvent event, GXILabelTag label) {
if (label.getId().equalsIgnoreCase("MyAppField")) {
        label.setPosition(new com.sabratec.util.GXPosition(3,30));
    }
}
```

Where `MyAppField` is the name of the mapped application field to reposition and 3,30 is the new position, in host units.

.NET

```
public override void onNewLabel(GXRenderEvent e, GXILabelTag label) {
if (label.getId() == "MyAppField") {
        label.setPosition(new com.sabratec.util.GXPosition(3,30));
    }
}
```

Where `MyAppField` is the name of the mapped application field to reposition and 3,30 is the new position, in host units.

Formatting Specific Fields

It is possible to display specific host fields in specific styles. For example, displaying the message line field in a large, red font. This is done using the style sheet (CSS) classes.

> **Note:** It is recommended to map the fields as application fields. Refer to the Instant Web Application Development Methodology section for general instructions on mapping fields to screen groups.

> **Note:** It is possible to carry out basic manipulation of fields using the Transformation Wizard, Text to Text transformation or Input Field to Text Field transformation (detailed in the Transformations).

▶ **To format a specific field:**

1   Create a new tag transformation and register it to the relevant screen group(s) or to all screens. Refer to **Creating a New Transformation**, **Applying a New Transformation to all Screens** and **Applying a New Transformation to a Screen Group**.

2   In the transformation class, add code that will reposition the field in the appropriate method. It will usually be onNewLabel (for output fields) or onNewTextField (for input fields). The following code demonstrates replacing a field with a hyperlink:

JSP

```
public void onNewLabel(GXRenderEvent event, GXILabelTag label) {
if (label.getId().equalsIgnoreCase("MyAppField")) {
label.removeAttribute("class"); //Needed for removing all css classes
        label.setAttribute("class","MyCSSClass");
}
}
```

Where MyAppField is the name of the mapped application field to format and MyCSSClass is the name of the CSS class that contains the required formatting.

.NET

```
public override void onNewLabel(GXRenderEvent e, GXILabelTag label) {
if (label.getId() == "MyAppField") {
label.removeAttribute("class"); //Needed for removing all css classes
        label.setAttribute("class","MyCSSClass");
    }
}
```

Where MyAppField is the name of the mapped application field to format and MyCSSClass is the name of the CSS class that contains the required formatting.

Replacing a Field's Text

It is possible to replace the original host text of a specific field with other text.

> **Note:** It is recommended to map the fields as application fields. Refer to the Instant Web Application Development Methodology section for general instructions on mapping fields to screen groups.

> **Note:** It is possible to carry out basic manipulation of fields using the Transformation Wizard, Text to Text transformation.

Create a new tag transformation and register it to the relevant screen group(s) or to all screens. Refer to **Creating a New Transformation**, **Applying a New Transformation to all Screens** and **Applying a New Transformation to a Screen Group**.

In the transformation class, add code that will replace the field's text. It will usually be `onNewLabel` (for output fields) or `onNewTextField` (for input fields):

JSP

```
public void onNewLabel(GXRenderEvent event, GXILabelTag label) {
if (label.getId().equalsIgnoreCase("MyAppField")) {
        String text = label.getContent().trim();
        if (text.equalsIgnoreCase("HostText1")) {
              label.setText("NewText1");
        } else if (text.equalsIgnoreCase("HostText2")) {
              label.setText("NewText2");
        }
        //...
}
```

Where `MyAppField` is the name of the mapped application field that its text is to be replaced, `HostText1, HostText2` are two original host texts and `NewText1, newText2` are two new texts.

.NET

```
public override void onNewLabel(GXRenderEvent e, GXILabelTag label) {
if (label.getId() == "MyAppField") {
        String text = label.getContent().Trim();
        if (text == "HostText1") {
              label.setText("NewText1");
        } else if (text == "HostText2") {
              label.setText("NewText2");
        }
        //...
    }
}
```

Where `MyAppField` is the name of the mapped application field that its text is to be replaced, `HostText1, HostText2` are two original host texts and `NewText1, newText2` are two new texts.

Replacing a Field with a Web Element, Adding a Web Element

It is possible to replace a specific field with a Web element such as a button, a hyperlink, an image etc.

> **Note:** It is recommended to map the fields as application fields. Instant Web Application Development Methodology section for general instructions on mapping fields to screen groups.

> **Note:** It is possible to carry out basic manipulation of fields using the Transformation Wizard, Text to Hyperlink transformation or Text to Image transformation or Text to Button transformation.

▶ **To replace a field with a Web element:**

1   Create a new tag transformation and register it to the relevant screen group(s) or to all screens. Refer to **Creating a New Transformation**, **Applying a New Transformation to all Screens** and **Applying a New Transformation to a Screen Group**.

2   In the transformation class, add code that will reposition the field in the appropriate method. It will usually be `onNewLabel` (for output fields) or `onNewTextField` (for input fields). The following code demonstrates replacing a field with a hyperlink:

JSP

```
public void onNewLabel(GXRenderEvent event, GXILabelTag label) {
if (label.getId().equalsIgnoreCase("MyAppField")) {
GXILinkTag myLink = ↵
com.sabratec.applinx.presentation.tags.html.GXHtmlTagFactory.instance().newLink("Software ↵
AG");
            myLink.setTarget("http://www.softwareag.com/");
            event.getScreenTagModel().replace(label, myLink);
    }
}
```

Where `MyAppField` is the name of the mapped application field to replace.

.NET

```
public override void onNewLabel(GXRenderEvent e, GXILabelTag label) {
if (label.getId() == "MyAppField") {
GXILinkTag myLink = ↵
com.sabratec.applinx.presentation.tags.html.GXHtmlTagFactory.instance().newLink("Software ↵
AG");
               myLink.setTarget("http://www.softwareag.com/");
               e.getScreenTagModel().replace(label, myLink);
        }
}
```

Where `MyAppField` is the name of the mapped application field to replace.

Manipulating Host Characters

It is possible to manipulate host characters, for example, removing unnecessary characters such as dots (.) or dashes (-), and replacing them with other text or HTML elements, etc.

> **Note:** It is possible to carry out basic manipulation of fields using the Transformation Wizard, Text to Text transformation.

▶ **To handle host characters:**

■ Create a new tag transformation and register it to the relevant screen group(s) or to all screens. Refer to **Creating a New Transformation**, **Applying a New Transformation to all Screens** and **Applying a New Transformation to a Screen Group**.

In the transformation class, add code that will manipulate the host characters as required. It will usually be `onNewLabel` (for output fields) or `onNewTextField` (for input fields). The following code demonstrates removing dashes (-):

JSP

```
public void onNewLabel(GXRenderEvent event, GXILabelTag label) {
if (label.getContent().indexOf("--") > -1) {
               String text = label.getText();
              text = com.sabratec.util.GXStringUtil.replaceAll(text,"-","");

               label.setText(text);
        }
}
```

.NET

```
public override void onNewLabel(GXRenderEvent e, GXILabelTag label) {
if (label.getContent().IndexOf("--") > -1) {
String txt = label.getText();
                txt = com.sabratec.util.GXStringUtil.replaceAll(txt,"-","");
                label.setText(txt);
    }
}
```

Manipulating Host Keys

It is possible to change the way host keys are displayed, add additional host keys and remove host keys.

▶ **To manipulate host keys**

■ Refer to the commented sample code in any ApplinX new application, UserHostKeysTagTransform.java/cs/vb.

Improving Transitions between Screens

Wrapping the `GXPlaceHolder` component in the template page with a Partial Page rendering capablility control such as a gx:div, will reduce the traffic between the web application server and the user client , since only the Instant part of the page will be transferred rather than the whole page. This will result in a slightly better performance and smoother transition between screens.

▶ **JSP:**

1 In the *Template.JSP* file, find the following control:

```
<gx:placeholder id="GXPagePlaceHolder">Design time page content</gx:placeholder> ↵
```

2 Place a `gx:div` tag around the `GXPagePlaceHolder`:

```
<gx:div id="instantPanel">
        <gx:placeholder id="GXPagePlaceHolder">Design time page ↵
content</gx:placeholder>
 </gx:div> ↵
```

3   Override the default behavior of each PF button on your keyboard. In *js/userExits.js*, change the `globalOnKeyDown` function to be of the following structure, where the if statement is before the `activateifexists` JavaScript code:

```
function globalOnKeyDown(gx_event){
        // use win.<SOMETHING> to access the page tags
        // for example: win.document.GX_form
        var win = gx_event.window;
        // if the key a PF key or ENTER
        if ((gx_event.keyCode>111 && gx_event.keyCode<124) || ↵
gx_event.keyCode==13 ){
                    // Update only the instantPanel part of the page
                    gx_updatePagePart('instantPanel');
        }
        // activate page scope function if exits
        activateIfExists(gx_event,gx_event.window.pageOnKeyDown); ↵
//gx_event.cancel();
        // for cancel the event
} ↵
```

▶ **.NET:**

1   In the *Template.master* file, find the following control:

```
<asp:ContentPlaceHolder ID="GXPagePlaceHolder" runat="server"/> ↵
```

2   Place a `div` tag around the `GXPagePlaceHolder`:

```
<div
        runat="server" id="instantPanel"> <asp:ContentPlaceHolder
        ID="GXPagePlaceHolder" runat="server"/>
</div> ↵
```

3   Override the default behavior of each PF button on your keyboard. In *js/userExits.js*, change the `globalOnKeyDown` function to be of the following structure, where the if statement is before the `activateifexists` JavaScript code:

```
function globalOnKeyDown(gx_event){
        // use win.<SOMETHING> to access the page tags
        // for example: win.document.GX_form
        var win = gx_event.window;
        // if the key a PF key or ENTER
        if ((gx_event.keyCode>111 && gx_event.keyCode<124) || ↵
gx_event.keyCode==13 ){
                    // Update only the instantPanel part of the page
                    gx_updatePagePart('instantPanel');
        }
        // activate page scope function if exits
        activateIfExists(gx_event,gx_event.window.pageOnKeyDown); ↵
//gx_event.cancel();
        // for cancel the event
} ↵
```

14   Emulation Behavior Tasks

ApplinX HTML Emulation provides a ready to use, fully functional Web emulation. It is part of ApplinX installation and is available in JSP and .NET. It is a thin-client Web application and uses JavaScript and HTML for configuration and fine-tuning.

To install a new HTML emulation refer to the Framework Manager or to the Eclipse Plug-in/Visual Studio documentation. The emulation template is based on the new_application Web application. It is possible to configure various features in the Framework Configuration Editor in the Emulation node. All the emulation features are relevant both for instant and generated pages (that inherit from GXDefaultLogicContext) unless otherwise displayed in the Emulation node in the Framework Configuration Editor. Detailed below are some tasks which require configuring a number of parameters.

Customizing the Background Check for Host Screen Changes

Use the user exits in *userExit.js*.

GXHostScreenChecker.hostScreenSeqCheckBeforeRefresh(isDirty) > returns boolean

This user exit method is used to perform additional queries for this feature. For example to check whether the data on the screen is saved. If it is not saved, return false to stop the browser from refreshing to prevent data loss.)

Example:

```
/**
 *    userExists.js     prompt user if the screen is dirty
 */
function GXHostScreenChecker.hostScreenSeqCheckBeforeRefresh(isDirty) {
      //check
      if (isDirty) {
            //ask & get answer from user...
                      alert("Host screen was updated. Data entered will be lost");
            return false; //stop refreshing
      }
      return true; //continue refreshing
}
```

GXHostScreenChecker.hostScreenSeqCheckSetTimer(ticks) > returns times

This user exit method is used to change the times for the checker timer. The user gets the current setting time and can return a different value. For example: the user finds 3, 6, 12, 24 too frequent, so in order to increase the time intervals he can return time*3 which results in 9, 18, 36, 72)

Example:

```
/**
 *    userExists.js     reset the timer
 */
function GXHostScreenChecker.hostScreenSeqCheckSetTimer(ticks) {
     return ticks*3;
}
```

Enabling the User to Control the Font Size

It is possible to enable the user to control the font size used in the Web application. The user will be able to increase/decrease the font size by clicking on the plus/minus buttons in the Web application. The font size is limited to be between 7 to 20 pixels.

▶ **To enable user control of the font size:**

1   Copy the plus/minus links from the page footer in the HTML emulation.

2   Set these links to call the `gx_increaseFontSize()` or the `gx_changeFontSize(<desired size in pixels>)` JavaScript functions and the `gx_decreaseFontSize()` JavaScript function.

Example:

```
<input type="button" value="+" onclick="gx_increaseFontSize(12);" />
```

3   Refer to **Configuring your Framework** and access the Framework Configuration Editor. In the **Instant** node, **Font size** field, select the **Dynamic by resolution** option from the drop-down list.

Refer to the API:

■ `gx_increaseFontSize`

■ `gx_decreaseFontSize`

■ `gx_changeFontSize`

Opening Multiple Web Sessions

A known problem of Web servers is that it is not possible to open multiple Web sessions (against the same Web application) from the same Web window (in a portal application for example). The ApplinX session is dependant on the web session, therefore it is problematic to open multiple host sessions using ApplinX framework. To overcome this problem ApplinX provides an ActiveX component, which creates a new IE process whenever a new Web window is required.

In *index.aspx/jsp* (in the emulation_template folder) uncomment the following line:

```
<OBJECT ID="gx_emulationComponent" ↵
CLASSID="CLSID:FE93BC5E-332E-41D7-9B36-EA36265998CA" ↵
CODEBASE="z_lib\GXOsApi.CAB#version=1,0,0,0"></OBJECT>
```

> **Note:** The following function already exists in the *index.aspx/jsp* file. Notice the use that the function does with the "gx_emulationComponent" object.

```
function connect(){
 var newUrl = ↵
"z_resourceReader.aspx?res=pages/z_openFull.htm&gx_page=gxfirstpage.aspx";
 if (window.gx_emulationComponent){
  // If the ActiveX was loaded use it to open a new
  // Internet Explorer window with its own IE process.
  gx_openNewBrowser(newUrl);
 }
 else{
  // If the ActiveX wasn't enabled open a window normally
  location.href=newUrl;
 }
}
```

Printing a Capture of the Host Screen

As in most terminal emulators, also ApplinX enables printing a snapshot of the current host screen. By executing a JavaScript function a popup screen appears enabling users to print the current screen. This window, unlike the Instant screen, does not include any changes (such as transformations) in the screen.

▶ **To enable users to print a capture of the host screen:**

■   Add a button/link to the application's Master page (*template.jsp/template.master.cs*). You can place it anywhere you see fit. Set it to call gx_printScreen().

Example

```
<input type="button" id="printButton" onClick="gx_printScreen();" />
```

Refer to the API:

▪ gx_printScreen

Enabling Sending Dup and FieldMark Characters to the Host

Dup and FieldMark are special mainframe characters. In instant and generated pages, a user can send to the host a not printable character in specific input fields.

▶ **To enable sending Dup and FieldMark characters to the host:**

1   Refer to **Configuring your Framework** and access the Framework Configuration Editor. In the **Emulation** node, select the **Support Dup and FieldMark host keys** check box.

2   In the **Keyboard mapping** node, map two keys, for example:

   Host action key: [dup]; Press a keyboard combination: CTRL+D

   Host action key: [fieldmark]; Press a keyboard combination: CTRL+F

3   You can also perform this mapping using JavaScript:

```
function pageOnLoad(){
    gx_engine.addKeyBoardMapping(
        GXAdditionalKey.CTRL,
        68 /*  Character 'd'  Ascii code */  ,
        gx_dup(),
        true);
}
```

4   When pressing CTRL+D (dup) the field will be marked with an asterisk ("*") and the cursor will move to the next field.

   When pressing CTRL+F (fieldMark) the field will be marked with a semi colon (";").

Refer to the API:

- ▪ `gx_fieldmark`

- ▪ `gx_dup`

15 Page Customization

Generating a Framework Page for a Screen

Refer to:

- Generating a JSP Page from a Screen
- Generating a .NET Page from a Screen

Creating Designed Web Pages

- Creating Designed Web Pages (JSP)
- Creating Designed Web Pages (.NET)

Creating Designed Web Pages (JSP)

Creating New JSP Pages

The ApplinX Framework for JSP provides an easy and simple way to develop a Web page based on host screen(s). This method is especially advantageous when the design of the pages is performed by a third party and all you have to do is just "bind" the HTML to ApplinX.

▶ **To create a new JSP page:**

1   Create a new JSP page and name it *<SCREEN_NAME>.jsp*. You can copy its content from *template.jsp*.

2   Import the tag library. The line can be copied from *template.jsp*.

3   Create a new java context class under *classes\contexts* and name it *<SCREEN_NAME>.java*. Copy the file content from *template.java*. Ensure that you change the class name to *<SCREEN_NAME>*.

4   Set the inheritance to `GXDefaultLogicContext/GXBasicContext` according to the development method.

5   Design the JSP page any way you like, or by pasting the third party HTML.

6   Declare `<gx:form>` opener and closer tags as the form tags.

7   If you require to bind an AppField to a dynamic tag:

- Add a prefix of gx: to the closer and end of the tag, or "/" closer for example, <gx:span id="span1"></gx:span> or <gx:input id="input1"/>

- Add an ID attribute: id="<APPFIELD_NAME>" to the tag which equals the AppField name you want the field to bind to.

- Make sure all the attributes are with "".

- The "class" attribute on the tag should be replaced with cssClass (reserved word in Java).

Any Java IDE that supports Web applications should auto-complete for the tag attributes. For any missing attributes or tags, you can expand the HTML tags library. The Tags library list can be viewed in the *classes\tags* folder, or *GXTags.tld* file under WEB-INF folder.

> **Note:** A gx tag should be well formed just like an XML node.

Tags Library Attributes

In order to work with ApplinX framework the new page context class should extend from `GXDefaultLogicContext` or `GXBasicContext`.

If you are using a context class for the JSP page, change the inheritance of the new context class, to `GXDefaultLogicContext` or `GXBasicContext`.

If you are not using a context class for the JSP page, set the context to `GXDefaultLogicContext` or `GXBasicContext`, by setting it in the `gx:html` : `<gx:html gx_context="contexts.GXDefaultLogicContext"`

How It Works

The filling-in of fields is performed at run time since each tag can get its content dynamically, which allows the ApplinX Framework engine to set content for the tags, without writing any code in them. The tag library allows you to view the JSP page on any other HTML editor, and redesign the page without worrying about the host logic.

Working with GXBasicContext (JSP)

If you decided to use `GXBasicContext`, change each link/button that performs any logic to "`<gx:input` … " or "`<g:a` …" and add an onserverclick attribute, with a function name (without "()") in the context class of the JSP page. The context class should contain a public function with this name without any parameters.

> **Note:** When inheriting from GXBasicContext, you should call the method gx_attach() in the load event of your code class.

For example: In `<myPage>.JSP`:

`<gx:html gx_context="contexts.myPage">`

…

`<gx:input id="myBtn" onserverclick="myBtn_click"/>`

`In classes\contexts/<myPage>.java`

```
Public void myBtn_click(){

}
```

You can use ApplinX framework "building blocks" gx functions in the triggered function. For example: `gx_doSubmitKeyLogic("[enter]")`, to send the field with an ENTER and jump to the next page.

Working with GXDefaultLogicWebForm (.NET)/ GXDefaultLogicContext (JSP)

If you decide to use `GXDefaultLogic`, and the designed page buttons/links should be PF keys, add `javascript:gx_SubmitKey('[<HOST_KEY>]')` for each button/link. Combining new server-side buttons along with `GXDefaultLogicWebForm` is also possible. When adding a new server-side button, ApplinX .NET does not interfere in the new event and it can be used for other purposes, or with paths/building blocks to perform any logic.

Creating Designed Web Pages (.NET)

Generating with ApplinX

Similar to the ASP/JSP Framework, ApplinX can generate for every screen a Web page in the .NET environment. You can implement this by employing the ApplinX Designer.

▶ **To add a file to the project (after generating it):**

1 Click on the project explorer top bar, click **show all files**.

2 Select the new generated file, right-click and select **include in project**.

3 You can choose to add a code behind file to the generated page if you select **Yes** for the prompt of the VS.NET.

4 After the new file is added to your project, change the inheritance of the code behind class to `GXDefaultLogicWebForm`.

How It Works

Unlike what occurs in the ASP/JSP framework, ApplinX uses the HTML control feature of ASP.NET, and generates an aspx page with standard HTML tags without any <%%> calls. The filling-in of fields is performed at run time since each generated tag is generated with the attribute `runat = "server"`, which allows the ApplinX framework engine to access the tags, without writing any code inside them. The generated HTML controls allow you to view the Web page on the .NET designer or any other HTML editor, and redesign the page without worrying about the host logic.

Creating a New .NET Web Form using VS.NET

The ApplinX ASP.NET framework provides another simple and easy way to develop a Web page based on host screen(s). This method is especially advantageous when the design of the pages is performed by a third party and all you have to do is just "bind" the HTML to ApplinX.

▶ **To create a new designed Web form using VS.NET:**

1   Create a new Web form and name it *<SCREEN_NAME>.aspx*.

2   Design it any way you like, or by pasting the third party HTML.

3   Declare *<form id="GX_form" runat = "server" >* as the main form, only if you would like to inherit from *GXDeafultLogicWebForm*.

4   Right-click on the VS.NET designer or add the attribute "Runat = "server"" to the tag for every HTML field you want to bind to ApplinX. Add an id attribute: "id="<APPFIELD_NAME>"" to the tag which equals the AppField name you want the field to bind to.

In order to work with ApplinX framework the new Web page should inherit from `GXDefaultLogicWebForm` or `GXBasicWebForm`.

If you are using a code behind for the Web page, change the inheritance of the new Web form, from `System.Web.UI.Page` to `GXDefaultLogicWebForm` or `GXBasicWebForm`.

If you are not using a code behind for the Web page, the inheritance is declared in the first line `<%@ Page … Inherits="GX…" %>`. You need to change the inheritance to `GXDefaultLogicWebForm` or `GXBasicWebForm`.

Working with GXBasicWebForm (.NET)

If you decide to use `GXBasicWebForm`, change each link/button that performs any logic to `runat = "server"`, and use the .NET designer to create a function behind the button, by double-clicking on it. You can use ApplinX framework "building blocks" gx functions to build the new page logic. For example: `gx_doSubmitKeyLogic("[enter]")`, to send the field with an enter and jump to the next page.

> **Note:** When inheriting from GXBasicWebForm, you should call the method gx_attach() in the load event of your code class.

For example:

In `<myPage>.ASPX:`

… `<gx:input id="myBtn" onserverclick="myBtn_click"/>`

In `<myPage>.aspx.cs`

`Public void myBtn_click(){`

```
}
```

Using Web Application Controls in Generated Pages

ApplinX frameworks support replacement of unprotected fields with a GUI element such as a combo box, radio-button or checkbox. Replacing an unprotected field can be achieved by using a transformation (typically in Instant pages) or directly in generated pages, using the following code:

.NET:

ApplinX .NET framework supports the following .NET controls:

```
<select id=<APP_FIELD_NAME>
     runat=server><option value="<HOST-VALUE>"> Display Value </option></select>
```

```
<gx:GXCheckBox id="<APP_FIELD_NAME>" runat=server
    checkedValue="<CHECKED-VALUE>" uncheckedValue="<UNCHECKED-VALUE>"
    />
<asp:RadioButtonList id="<APP_FIELD_NAME>" runat="server">
<asp:ListItem Value="<HOST-VALUE>">Display value</asp:ListItem>
<asp:ListItem Value="<HOST-VALUE2>">Display value2</asp:ListItem>
</asp:RadioButtonList> ↵
```

The `gx:Checkbox` tag requires users to add a tag registration statement at the top of the page as follows: `<%@ Register TagPrefix="gx"`
`Namespace="com.sabratec.dotnet.framework.web.controls" Assembly="GXDotnet" %>`

> **Note:** When adding the attribute `multiple="true"` the combobox will be mapped to the values of a multiple field.

> **Note:** Only checkbox GUI elements need to be configured in the ApplinX Designer.

JSP:

ApplinX Framework for JSP supports the following JSP tags:

```
<gx:select
     id=<APP_FIELD_NAME>><option value="<HOST-VALUE>"> Display Value ↵
</option></gx:select>
```

```
<gx:checkbox id="<APP_FIELD_NAME>" checkedValue="<CHECKED-VALUE>" ↵
uncheckedValue="<UNCHECKED-VALUE>"
    />
<gx:radioButtonList id="<APP_FIELD_NAME>" runat="server">
<gx:radioButton value="<HOST-VALUE>">Display value</gx:radioButton>
 <gx:radioButton value="<HOST-VALUE2>">Display value2</gx:radioButton>
 </gx:radioButtonList>
```

> **Note:** When adding the attribute `multiple="true"` the combobox will be mapped to the values of a multiple field.

Partial Page Rendering

Partial Page Rendering (PPR) is a feature that allows a part of a page to be redrawn rather than having to reload the entire page. Partial Page Rendering offers significant benefits as it improves application performance as well as provides more direct feedback when users perform actions. You can add server side buttons/links and perform logic on the target server function, and as a result update only a part of the page. Do not use this feature if the action triggers navigation to a different web page.

Do not call getResponse.sendRedirect() (JSP) /Response.Redirect (.NET), gx_handleHostResponse in the target function.

Typical use: Paging, sorting, collecting aggregated data from a few screens into the same web page, and more.

1. Add a button/link which performs a server side action (refer to **Creating a Button/Hyperlink for Submitting a Host Key**).

2. Create a wrapping tag to the web page in the area you want to update:

   JSP:

   ```
   <gx:div id="tableArea">
   …
   </gx:div>
   ```

.NET

```
<div id="tableArea" runat="server">
…
</div>
```

3. Add the following JavaScript code to the link/button:

```
… onclick="gx_updatePagePart('tableArea')"
… onserverclick="Btn_Clicked" …
```

Creating a Button / Hyperlink for Submitting a Host Key

▶ **To create a button / link for submitting a host key (JSP):**

1   Add a button or a hyperlink to a page, with the attribute

```
onserverclick="<name_of_server_side_function>":
```

```
<gx:a id="mylink" onserverclick="mylink_ServerClick">Top</gx:a>
```

2   In the Code-behind of the page, add an "onServerClick" function of the following form:

```
public void mylink_ServerClick(){
    gx_doSubmitKeyLogic("[pf3]"); } ↵
```

▶ **To create a button / link for submitting a host key (.NET):**

1   Add a button or a hyperlink to a page, with the attribute

```
runat="server": <a href="" runat="server" id="myLink"
onserverclick="myLink_ServerClick">
```

2   In the code-behind of the page, add an "onServerClick" function of the following form:

```
private void myLink_ServerClick(object sender,
        System.EventArgs e) { gx_doSubmitKeyLogic("[pf3]");
}
```

▶ **To create a button / link for submitting a host key (JavaScript):**

■   Add a button or a hyperlink to a page with the attribute:

```
onclick="gx_SubmitKey([keyValue])":
<a href="#" id="myButton" onclick="gx_SubmitKey([pf3])">Back</a>
```

Sample code can be found in the Composite demo: BrowseProposals.jsp/cs containing links that submit [pf4] and [pf3].

Refer to the API:

- gx_SubmitKey
- gx_SubmitKeyInPos
- gx_SetCursorPos
- gx_systemRequest
- SubmitCustomKey

Creating a Button / Hyperlink for Executing a Path Procedure

▶ **To create a button / link for executing a navigation path (JSP):**

1   Add a button or a hyperlink to a page, with the attribute
    onServerClick="<name_of_server_side_function>":

    ```
    <gx:a id="myLink" onserverclick="myLink_ServerClick">Top</gx:a>
    ```

2   In the code-behind of the page, add an "onServerClick" function of the following form:

```
public void myLink_ServerClick(){
try {
GXPathRequest req = new GXPathRequest("<path_name>");
      req.addVariable("<PathInputVariable1>","<value1>");
      req.addVariable("<PathInputVariable2>","<value2>");
// ... add additional input values
      getGXSession().executePath(req);
 gx_handleHostResponse();
 } catch(GXGeneralException err){
 //handle error
}
}
```

▶ **To create a button / link for executing a navigation path (.NET):**

1   Add a button or a hyperlink to a page, with the attribute

    ```
    runat="server": <a href="" runat="server" id="myLink">
    ```

2   In the code-behind of the page, add an "onServerClick" function of the following form:

```
private void myLink_ServerClick(object sender,
        System.EventArgs e) {
try {
GXPathRequest req = new GXPathRequest("<path_name>");
      req.addVariable("<PathInputVariable1>","<value1>");
      req.addVariable("<PathInputVariable2>","<value2>");
// ... add additional input values
      gx_session.executePath(req);
gx_handleHostResponse();
} catch(GXGeneralException err){
 //handle error
}
}
```

▶ **To create a button / link for executing a navigation path (JavaScript):**

■   Add a button or a hyperlink to a page with the attribute:

```
onclick="gx_ExecPath('<path name>')"
<a href= "#" id="myButton" onclick="gx_ExecPath('gotoMainMenu')">Top Men</a>
```

Sample code can be found in the Composite demo Web application: login.jsp/cs – containing a button that executes a login path.

Collect all Modified Page Fields into an ApplinX Request

The framework is able to perform automatic actions against the host, by using a simple API. One of the common tasks is to collect all the modified fields from the Web page into a GXSendKeyRequest / GXPathRequest object, add/remove fields from it, and send it to the host.

▶ **To collect all modified page fields into an ApplinX request (JSP)**

■   import com.sabratec.applinx.baseobject.*;

```
 // collect all modified fields from the page, along with [enter] key
GXSendkeysRequest req = gx_prepareSendKeysRequest("[enter]");

// collect all modified fields from the page that
// matches the current screen, along with the specified path name
//GXPathRequest req = gx_preparePathRequest("<PATH NAME>");

// modify/add a field to the request
req.addInputField("<SOME FIELD>","<SOME VALUE>");

// send the request to the host
getGXSession().sendKeys(req);

// for path execution
// getGXSession().executePath(req);

if (getGXSession().getScreen().getName().equals("<SOME SCREEN NAME>")){
        // performs page actions
}
else{
        // jump to the relevant page
        gx_handleHostReponse();
}
```

▶ **To collect all modified page fields into an ApplinX request (.NET)**

■   Using com.sabratec.applinx.baseobject;

```
 // collect all modified fields from the page, along with [enter] key
GXSendkeysRequest req = gx_prepareSendKeysRequest("[enter]");

// collect all modified fields from the page that
// matches the current screen, along with the specified path name
//GXPathRequest req = gx_preparePathRequest("<PATH NAME>");

// Modify/add a field to the request
req.addInputField("<SOME FIELD>","<SOME VALUE>");

// send the request to the host
gx_session.sendKeys(req);

// for path execution
//gx_session.executePath(req);

if (gx_session.getScreen().getName() == "<SOME SCREEN NAME>"){
        // performs page actions
}
else{
        // jump to the relevant page
```

```
                        gx_handleHostReponse();
}
```

Exporting Data to an MS Office Application (Excel, Word)

ApplinX data can be easily exported to an MS Office application, such as an Excel spreadsheet as detailed in this example. 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 JSP file (For example, if you want to add a border to a table, in the *excel.jsp* file, add to the table tag: `<table border=1>`).

> **Note:** The solution provided below is an applicative solution. The following solution can be implemented to export data in other formats (such as CSV or Word documents). An additional example can be found in the Composite Demo, in which the HTML table is exported to an Excel document.

▶ **To export an ApplinX Table to an Excel Spreadsheet (JSP):**

1   Define a Web page and a java class bound to it. Add the desired gx tags for the Excel data. For example: `<gx:table id="myTable"/><gx:span id="excelData"/>`

2   In the java class, extend `GXBasicContext`. In the function `gx_onLoad` define the page's content type as Excel:

```
getResponse().reset();
getResponse().setContentType("application/vnd.ms-excel");
getResponse().addHeader("Content-Disposition","attachment; filename=\"" +
gx_table.getName() + ".xls\"");
```

3   Attach to ApplinX: `gx_attach();`

4   Get the ApplinX host screen data:

```
GXITable gx_table = getSession().getScreen().getTables()[0];
```

5   Fill the gx tags with the retrieved data: getTagAccesor( ).setTagContent("excelData",xlStr); getTagsAccesor( ).setTagTable("excelTable,gx_Table);

6   This code is generic and the page can be invoked from any other page requiring export to Excel.

Sample code can be found in the Composite Demo Application:

■ WEB-INF\classes\contexts\excel.java – Changing the HTTP response content-type to Excel and filling the HTML table.

■ \excel.jsp – returning the spreadsheet by invoking excel.java.

■ \BrowseProposals.jsp – invoking excel.jsp upon clicking an image.

▶ **To export an ApplinX Table to an Excel Spreadsheet (.NET):**

1 Define a page that extends `GXBasicWebForm`. Add a table to the page (with an runat="server" attribute). In the function `Page_Load` define the page's content type as Excel:

```
Response.ContentType = "application/vnd.ms-excel";
 Response.AddHeader("content-disposition","attachment; filename=\"" +
 gx_table.getName() + ".xls\"");
```

2 Attach to ApplinX: `gx_attach();`

3 Get the ApplinX host screen data:

```
GXITable gx_table = gx_session.getScreen().getTables()[0];
```

4 Bind the DataTable to the DataGrid: `DataGrid1.DataSource = dt; DataGrid1.DataBind();`

5 This code is generic and the page can be invoked from any other page that it is required to export to Excel.

Sample code can be found in the Composite Demo Application: \excel.aspx – binding an ApplinX table to the Excel spreadsheet and returning it. \BrowseProposals.aspx – invoking excel.aspx upon clicking an image.

Building an External Login Page

An external login means a login Web page that is not connected to the host when the page is loaded. It allows displaying the typed user name as the ApplinX session ID within the Designer.

▶ **To create such a page:**

1 Create a login path using ApplinX. This path should login to the host application (by supplying the user ID and password) and navigate from the first screen of the host application to the main menu screen of the application (or any other screen that appears after logging in).

2 Set the code class for your login page to inherit from `GXBasicWebForm` `(.NET)/GXBasicContext(JSP)`.

3 Initialize the session configuration object with the typed user name and (optional) the typed password (for securing the ApplinX session from other users).

4 Use the framework building block `gx_connect` to create a new session.

5 Execute the path method of the base object.

6   Add error handling as needed, for example: disconnect if login failed.

7   Navigate to the next matching page (`gx_handleHostResponse`).

For code sample see Composite Demo Application: common/Login.aspx / Login.aspx.cs - .NET
Login.jsp / WEB-INF/classes/contexts/common/Login.java – JSP

Collecting Data from Multiple Host Screens

It is a common task to collect data from multiple host screens. ApplinX enables you to do this by using a path procedure that navigates through the host screens and collects different types of data to be retrieved as procedure's output. The retrieved data can then be used either in a Web Service/Procedure Client or be bound to web controls defined in an ApplinX framework based web page.

▶

1   Create a Path procedure which navigates through several host screens and map host data as procedure output.

2   Associate the procedure you created with a procedure group.

3   The procedure group can be used in one of the following ways:

   1. Web service – wsdl file can be found at the following URL: http://<ApplinX Server machine>:2380/wsstack/services/<ApplinX Application Name>.<Procedure group name>?wsdl

   2. Procedure Client – It is possible to generate a code client into your own .NET/Java project. The generated code will include:

   ◾ Classes for ApplinX defined types (Data Structure entities) used by the procedure group methods (associated procedures).

   ◾ Methods that execute the procedure group methods (associated procedures).

   ◾ A service that handles ApplinX server connectivity.

Binding Procedure Outputs to an ApplinX Framework Based Web Page

It is possible to bind Procedure outputs to several controls on a web page so that the appropriate controls are displayed on the web page. In order to be able to do this with minimum code, follow these steps:

1. Make sure your procedure's outputs are well defined for web page binding:

   a. Text and Text array outputs will easily bind to a text control (Input or Span for example) in a web page providing this control has the appropriate ID attribute (see examples below).

   b. Data structure arrays can be used to bind data to a Table control on a web page providing that the table has the same ID as the array name and contains controls (Input, Span, TD, SELECT tags) with IDs similar to the data structure's attribute names (see example below).

2. Associate the procedure with a Procedure Group in order to be able to generate a Procedure Client into your project (if you haven't already done so).

3. If you have made changes to your procedure inputs/outputs, re-generate the Procedure Client into your project in order for the changes to take effect.

4. Create a Fill method that executes the procedure and fills the Form with the data provided by the procedure's response.

Example

The following example displays the procedure output (example: myProcedure):

The procedure output contains an Array (named Customers) of CustomerDetails type which we defined earlier. This type has several attributes (first name, last name….). It also contains a single Text output (named MSG) used to retrieve messages from the host during execution. Assuming we wish to display the Customers Array in a table and the host message in a SPAN tag, this is what the controls look like in the JSP/ASPX page:

JSP

```
<gx:span id="MSG"></gx:span>

<gx:table id="Customers" clas="gx_tbl" cellspacing=...>
   <tr>
       <th>ID</th>
       <th>First Name</th>
       ......
       <th>Birth Date</th>
   </tr>
   <tr>
       <td><gx:span id="customerID"></gx:span>/td>
       <td><gx:span id="firstName"></gx:span></td>
```

```
      .......
      <td><gx:span id="birthday"></gx:span></td>
   </tr>
</gx:table>
```

.NET

All relevant controls must have a "runat='server'" attribute

```
<span id="MSG" runat="server"></gx:span>

<table id="Customers" runat="server" clas="gx_tbl" cellspacing=...>
   <tr>
      <th>ID</th>
      <th>First Name</th>
      ....
      <th>Birth Date</th>
   </tr>
   <tr>
      <td><span id="customerID" runat="server"></span>/td>
      <td><span id="firstName" runat="server"></span></td>
      .....
      <td><span id="birthday" runat="server"></span></td>
   </tr>
</table>
```
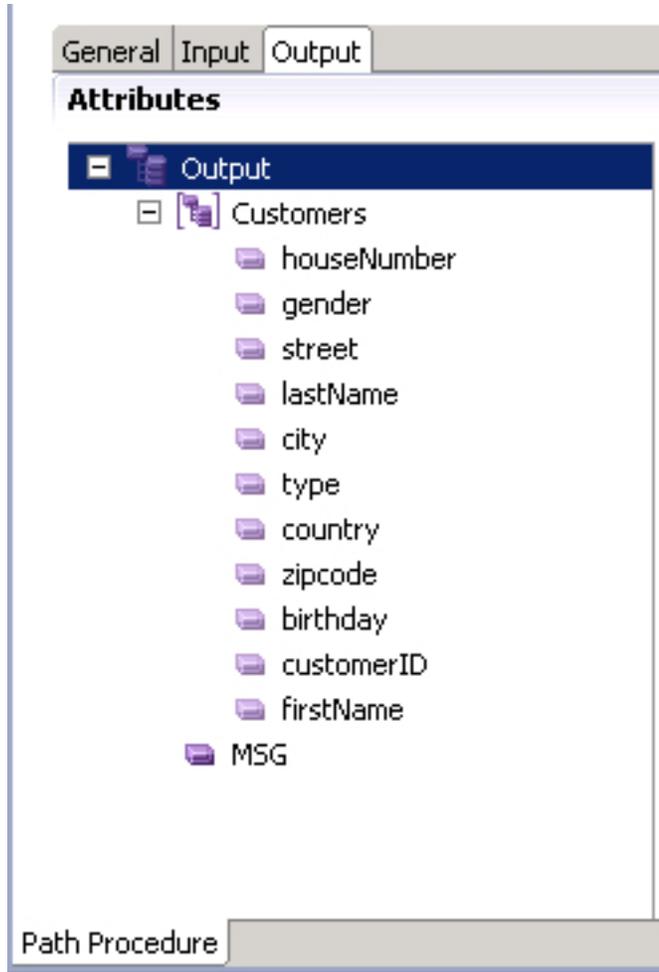
After associating the procedure (in the above example: <myProcedure>) with a procedure group (in the example: <myGRP>) and generating a Procedure Client, execute the procedure client and fills the web page with relevant data:

In the page's code behind (located at \WEB-INF\classes\contexts\), add a function (example: myFillMethod), executing the procedure client and filling the page with the path's output:

Fill a Form from an open ApplinX Web Session:

JSP

```
public void myFillMethod (){
 <myProcedure>Request req = new <myProcedure>Request();
 <myProcedure>Response res = ↵
(<myProcedure>Response)getGXSession().executeProcedure(req);
   gx_fillForm(res);
}
```

.NET

```
public void myFillMethod (){
 <myProcedure>Request req = new <myProcedure>Request();
 <myProcedure>Response res = (<myProcedure>Response)gx_session.executeProcedure(req);
   gx_fillForm(res);
}
```

When Should my Fill method be executed?

Determining when to execute the Fill method depends on the Page behavior. This functionality can be added to a method, triggered by a button's onClick event to fill data on the screen. We could override the default gx_fillForm method to add some data to the screen when the page is initially loaded, by executing and retrieving data for other host screens. These are just two examples. This method can be executed by any server side page event.

Note:

■ Make sure your fill method doesn't collide with the page's gx_fillForm method. If you execute your method before the gx_fillForm method has been executed some of data may be lost. In this case, check screen fields and output field names. If you find matches, change one of them.

■ Output names are case sensitive. Make sure web controls are named appropriately , otherwise, they will not be filled.

Updating Data in Multiple Host Screens

It is a common task to update/enter data in multiple host screens using a single Web page. ApplinX enables you to do this by using a path procedure for updating the data on the screens (refer to Creating a Path Procedure), and calling your own server side functions to bind the path's input variables to the fields defined in each of the host screens. This data is often presented using Web tabs.

▶

1   Create/Record a Path Procedure which updates/enters data in multiple host screens.

2   In the JSP/ASPX file, add controls to allow users enter the input data required by the procedure. Consider the following:

   ■ gx:input controls, with an identical ID as the input attributes name, will make it easy for you to track which JSP/ASPX control should be assigned to which input variable.

   ■ Using Data structures will require you to generate a procedure client in order for the framework to properly recognize the structure defined in APX repository.

3   Create a server side execution method and a Link/Button on the JSP/ASPX page that will trigger the function(refer to Creating a Button / Hyperlink for Executing a Path Procedure). Refer to ModifyProposal1.java/aspx.cs.

▶ **To update data in multiple host screens (JSP):**

1   Create a Web page (JSP) for the first screen of the path procedure, containing fields representing each input field in the procedure, for example:

```
<gx:input id="StockValueGroup" type="text" maxlength="1" size="1"/>
```

2   In the page's code behind (located in \WEB-INF\classes\contexts\), create a doSave method, executing the path procedure with the procedure's input:

```
public void doSave(){ GXPathRequest req = new GXPathRequest ();
        req.addVariable("Input1", ↵
getTagsAccesor().getTagContent("StockValueGroup"));
        req.addVariable("Input2","Input2Value");
       req.addVariable("Input3","Input3Value"); ... ... ... GXPathResponse res =
        getGXSession().executePath(req); gx_handleHostResponse(res); } ↵
```

▶ **To update data in multiple host screens (.NET):**

1   Create a Web page (JSP) for the first screen of the collection path, containing fields representing the fields appearing in all accumulated host screens, for example:

```
<gx:input id="StockValueGroup" type="text" maxlength="1" size="1 "runat="server">
```

2   In the page's code behind (.aspx.cs or .aspx.vb), create a doSave method executing the path procedure with the procedure's input:

```
public void doSave(object sender,EventArgs args)
{
   try
   {
       GXPathRequest req = new GXPathRequest ();
       req.addVariable("Input1",  StockValueGroup.Text));
       req.addVariable("Input2","Input2Value");
       req.addVariable("Input3","Input3Value");
       ... ... ...
       GXPathResponse res = gx_session.executePath(req);
       gx_handleHostResponse(res);
   }
   catch (GXGeneralException ge)
   {
           gx_handleSessionError();
   }
}
```

Sample code can be found in the Composite demo Web application. ModifyProposal1.aspx / ModifyProposal1.aspx.csis a page which contains tabs displaying data collected from multiple host screens, with the ability to update the data (online only).

Activating a Server Side Function from JavaScript

In some cases it is required to activate a server side function after performing a number of client side actions such as validation, hierarchical-menus, etc. In order to do so, perform the task detailed below.

▶ **To activate a server side function from JavaScript (JSP):**

■   `gx_postBack('<SERVER FUNCTION NAME>');`

▶ **To activate a server side function from JavaScript (.NET):**

1   Add a hidden button to the page:

```
<input type="button" runat="server" id="MyBtn" style="display:none"
onserverclick="<SERVER FUNCTION NAME>">
```

2   The command to execute it from JavaScript:

```
GXBrowserUtil.getElement("MyBtn").click();
```

Mapping Keyboard Keys to User Actions in Individual Pages

Page specific actions can be mapped to a keyboard key (or to a combination of two keyboard keys) using the following JavaScript in the page.

```
function pageOnLoad(){
gx_engine.addKeyBoardMapping(GXAdditionalKey.<the additional key>, GXKeyCodes.<the ↵
key to which the event is attached>,myJSFunction,<determines whether to override ↵
existing mapping>,[myJSBoolFunc]);
}

function myJSFunction(){
          // enter your code
}

// this function is optional and can be left out - See examples
function myJSBoolFunc(){
 if (…)
```

```
  return true;  // myJSFunction will not be executed
 else
  return false; // myJSFunction will be executed
}
```

Examples

```
// Map [+] to the JavaScript function addClient();
// addClient function will override any default functionality
// associated with the [+] key
gx_AddKeyBoardMapping(GXAdditionalKey.NONE,GXKeyCodes.PLUS,addClient,true);

//Map shift+ESC -> javaScript:logoffUser();
gx_AddKeyBoardMapping(GXAdditionalKey.SHIFT,GXKeyCodes.ESC,logoffUser,false);

//map ctrl+TAB -> javaScript:printPage();
gx_AddKeyBoardMapping(GXAdditionalKey.CTRL,GXKeyCodes.TAB,printPage,false);

// Map [ENTER] to the JavaScript function dropClient();
// dropClient will be performed if Javascript function confirmFunc
// returns false
gx_AddKeyBoardMapping(GXAdditionalKey.NONE,GXKeyCodes.ENTER,dropClient,true,confirmFunc);
```

Available Constants

Additional Keys

GXAdditionalKey.NONE

GXAdditionalKey.CTRL

GXAdditionalKey.ALT

GXAdditionalKey.SHIFT

GXKeyCodes.BACKSPACE	GXKeyCodes.HOME	GXKeyCodes.F3
GXKeyCodes.TAB	GXKeyCodes.LEFT	GXKeyCodes.F4
GXKeyCodes.ENTER	GXKeyCodes.UP	GXKeyCodes.F5
GXKeyCodes.SHIFT	GXKeyCodes.RIGHT	GXKeyCodes.F6
GXKeyCodes.CTRL	GXKeyCodes.DOWN	GXKeyCodes.F7
GXKeyCodes.ALT	GXKeyCodes.PLUS	GXKeyCodes.F8
GXKeyCodes.CAPSLOCK	GXKeyCodes.INSERT	GXKeyCodes.F9
GXKeyCodes.ESC	GXKeyCodes.DELETE	GXKeyCodes.F10
GXKeyCodes.PAGEUP	GXKeyCodes.F1	GXKeyCodes.F11
GXKeyCodes.PAGEDOWN	GXKeyCodes.F2	GXKeyCodes.F12

GXKeyCodes.END		

Key Codes

Refer to the API:

■ Keyboard Mapping

Handling the Screen Locker on the Page Level

The purpose of a screen locker is to indicate to the user by means of a message, that the application is processing his request, and blocks him from interfering with the current process by repressing a button/link or keyboard PF/ENTER. The Screen Locker feature is typically implemented for the whole application (Refer to **Activating the Screen Locker**). Sometimes it is necessary to initiate a screen-lock or disable the screen locker for a specific page.

For example, in the following code, the page contains a link to the Software AG web site, and the Href attribute executes a JavaScript function: openWin(). If we do not disable the Screen Locker, the application will be locked and the user will not understand the reason that it is locked. However, as the window opened (in this example, the Software AG web site) is not related to the application, the screen will not become unlocked. Therefore, in such a case, disabling the screen locker, allows the user to carry on working. First, ensure that you have configured the relevant parameter in the Framework Configuration Editor.

```
<script>
      function openWin(){
          gx_disableScreenLocker();
          window.open('http://www.softwareag.com,'newWin',
                      'width=300,height=300,top=10,left=0');
      }
</script>

<a href="javascript:openWin();">Software AG</a>
```

Refer to the API:

gx_lockScreen(): used to lock the web page so that no additional host keys/links/ buttons can be pressed until the page is unlocked.

gx_unlockScreen(): used to unlock a web page.

gx_disableScreenLocker()/gx_enableScreenLocker(): used to disable/enable the screen locker. Designed to lock the screen while the client is waiting for a new application screen.

Navigating between Input Fields

By default the framework offers using the Up arrow key, Down arrow key and Tab key to navigate between inputs. It is possible to define additional navigation options such as moving to the first or last element in a page, moving to the next input field, moving to the first or last input field in the screen.

The following example moves the cursor to the next input element whenever the right arrow key is pressed on the keyboard. Add the following function to any generated page or even to the master page (*template.jsp /template.master.cs*) in your web application.

```
function pageOnKeyDown(gx_event){
 // activate page scope function if exits
 activateIfExists(gx_event,gx_event.window.pageOnKeyDown);

 if (gx_isValidInputElement(gx_event.element)){
     if (gx_event.keyCode== GXKeyCodes.RIGHT){
         gx_jumpToNextInput(gx_event.element);
     }
 }

}
```

Refer to the API:

- gx_home
- gx_end
- gx_newLine
- gx_jumpToNextInput
- gx_jumpToPrevInput

Retrieving Browser Information

The web application is sometimes required to be cross browser compatible. While most JavaScript code runs well on all browsers, there are some subtle differences between how different browsers handle the same JavaScript code. Therefore it is important to determine the browser used. The ApplinX JavaScript engine provides several functions to retrieve the browser information.

Example for dealing with cross-browser JavaScript

```
function myFunction(){
 if (GXBrowserUtil.isIE()){   // is the client using Internet Explorer
if (GXBrowserUtil.isIE7()){        // is it IE version 7 (or above)?
   doThis();
}
else{      // it's IE6 (or lower)
 doThat();
}
}
if (GXBrowserUtil.isMozilla ()){  // is the client using Firefox?
  doSomethingElse();
}
}
```

Refer to the API:

▪ GXBrowserUtil.isIE

▪ GXBrowserUtil.isIE7

▪ GXBrowserUtil.isMozilla

Validating your Data

ApplinX Web enablement sends information to the host screen. Therefore, the host application validation continues to validate the data in the same way as when using a standard emulation. However, to enhance existing validations and to be more user friendly, it is possible to use Web page validations, which validate the data before sending the information to ApplinX and then to the host. A typical example for such a case is when merging several host screens into a single web page.

Additional usages include:

▪ Required-fields validation.

▪ String format validation (email, date etc.)

▪ Numeric/alphanumeric field validation.

GXValidator represents an array of validators, used to validate the form's data before submitting it to the host.

The following example is automatically added to every JSP/ASPX page generated from ApplinX (as a commented example). This validator checks that "Field_A" has a value. Otherwise, it returns an error message.

```
var pageValidator = new function(){
 this.validateField = function(inputField){
  if (inputField.name == "FIELD_A"){
   if (inputField.value == ""){
    return "Field cannot be empty";
   }
  }
  // ...
 }
}

function pageOnLoad(){  // register validator function
 GXValidator.registerValidator(pageValidator);
}
```

The form is validated when submitted. The returned message is reflected in the "errormsg" div located in the master page (template.jsp /template.master.cs). Once a validator indicates invalid data, an error message will appear in the web page and the form will not be submitted.

Refer to the API:

■ GXValidator.registerValidator

■ GXValidator.clearValidators

Handling Web Application Windows using the gx_windows Object

When working with pop-up windows, the interaction between the pop-up window and the parent window needs to be defined, in order to ensure continuous and smooth workflow when working with windows. For example, ApplinX framework uses the gx_window object to automatically open pop-up windows, reflecting host application window screens. In addition it is also possible to open windows to perform other actions which are not related to the application, such as opening a window or a different web page. The JavaScript gx_window object encapsulates the web page window object with additional ApplinX functionality enabling developers to perform JavaScript actions without disrupting the ApplinX JavaScript workflow. When using ApplinX frameworks, it is recommended to use the gx_window object.

The following example demonstrates how to open a pop up window from the main window and fill in an input element on the main page from data taken from the pop up page. In addition, the child window JavaScript function handles the screen locker of the parent page.

Main window HTML and Script:

```
<script>
     function getData(){
       gx_window.open("myData.jsp",200,300,0,0) ;

  //  .NET
  // gx_window.open("myData.ASPX",200,300,0,0);
     }
     </script>
<input id="myInput"  value=" "/>
<a href="#" onclick=" getData ();">open Win</a>
```

Pop up Window HTML and Script

```
function setMainWindowInput(){
          // Set the value of an Input on the main window
                gx_window.getOpener().
                    gx_window.setField("myInput","someValue");
                // Unlock the main window before closing the pop up
         gx_window.getOpener().gx_unlockScreen();
                // Close the pop up window
         gx_window.close();
   }

<a href="#" onclick=" setMainWindowInput ();">close Win</a>
```

Refer to the API:

- gx_window.open
- gx_window.resizeTo
- gx_window.moveTo
- gx_window.setField
- gx_window.close
- gx_window.getOpener

Working with Cookies

You may want to store user information and adjust the ApplinX Web Application to each user's needs. This information can be stored in cookies. ApplinX JavaScript Engine provides developers with easy to use JavaScript functions to set and get Cookie values. You can create new cookies and use the values to further meet user requirements. The ApplinX HTML Emulation template Application makes use of cookies to store the user's last choice of font-size, style sheet preference and screen resolution.

The following example will demonstrate how you can check if the user has a preferred style sheet. If not, a default value will be set and the document style sheet reference changed. The following code can be added to the instant page in the Emulation template application:

```
function pageOnLoad(){
   //Check if cookie doesn't exist
   if (!gx_getCookie(GXLAFHandler.COLOR_CSS_COOKIE)){
  // set the cookie value, so that the next page that loads
  // automatically uses the styles_black css file
      gx_setCookie(GXLAFHandler.COLOR_CSS_COOKIE,
                               "css/styles_black.css");
  // Set the current page to use that File as well
      document.styleSheets[0].href =
          gx_getCookie(GXLAFHandler.COLOR_CSS_COOKIE);
   }
}
```

Refer to the API:

- gx_getCookie

- gx_setCookie

Working with JavaScript User Exits

The user exits file is located in the js directory of your web application. All the global events registered in the globalOnLoad function allow users to add JavaScript functionality to the default framework behavior. Any code placed in these global functions will effect the whole application and therefore must be handled with caution. Each global function contains a the following line: activateIfExists(gx_event,gx_event.window.functionName); This line of code will execute the user function, functionName, if indeed the function exists on the page.

For Example: Assuming you have a text area on your web page, whenever you press the enter key, the page will be submitted to the host rather than just add a new line to the text area. Adding the following function to a generated JSP/ASPX, that contains a text Area input control, in the ApplinX web application will cause this function to automatically execute whenever a keyboard key is pressed.

```
// When a keyboard key is pressed check if "enter" was pressed and if the event ↵
originated in
// "myTextArea". If so, Prevent the form submitting by canceling the event and ↵
manually add newline
// to the textArea value
function pageOnKeyDown(gx_event){
 if (gx_event.keyCode == GXKeyCodes.ENTER &&
                     gx_event.element.id == "myTextArea"){
     gx_event.element.value += "\r\n";
```

```
     gx_event.cancel();
 }
}
If you wish to implement this behavior to all text areas in your web application ↵
add the following code to the globalOnKeyDown function in the userExits.js file
// When a keyboard key is pressed check if "enter" was pressed and if the event ↵
originated in a
// TextArea. If so, Prevent the form submitting by canceling the event and manually ↵
add newline
// to the textArea value
function globalOnKeyDown (gx_event){
 // activate page scope function if exits
 activateIfExists(gx_event,gx_event.window.pageOnKeyDown);
 if (gx_event.keyCode == GXKeyCodes.ENTER &&
                      gx_event.element.tagName == "TEXTAREA"){
     gx_event.element.value += "\r\n";
     gx_event.cancel();
 }
}
```

Refer to the API:

▪ User Exits

Retrieving HTML Objects using gx_getElement

As ApplinX works with HTML frames, you need to be sure that you work with the current HTML document object model (DOM). It is therefore recommended to use the gx_getElement function (instead of the window.getElement JavaScript function.

The following example demonstrates how to retrieve a message received from the host and launch it as an alert message when the page is loaded. Add the following to any generated page:

```
function pageOnLoad(){
// Get the message from the "Message" SPAN
 var msg = gx_getElement("MessageSpan").innerHTML;
 // If the message contains something other than spaces (&nbsp)
 if (msg.replace(/&nbsp/g, "")!=""){
 // launch an alert message and replace any &nbsp with space
  alert(msg.replace(/&nbsp/g, " "));
 }
}
```

Using the Calendar Component in Generated Pages

AplinX provides the functionality to add a calendar component. In Instant pages this is done using transformations and in generated pages this is done using JavaScript code as demonstrated in the following examples.

In this example, when choosing a date from the calendar, it will be populated in a 'MM-dd-yyyy' format. .e.g. 12-31-2008.

```
<input maxlength="10" id="birthDate" />
<a href="#" onclick="gx_showCalendar('Select a Date','birthDate','MM-dd-yyyy');">
<img src="images/calendar.gif"/>
</a>
```

In this example, when choosing a date from the calendar, it will populate three input fields, each having its own format:

```
Day:    <input maxlength="10" id="day" />
Month:  <input maxlength="10" id="Month" />
Year:   <input maxlength="10" id="Year" />
<a href="#" onclick="gx_showCalendar('Select a ↵
Date','Month','MM','year','yy','day','dd');">
<img src="images/calendar.gif"/>
</a>
```

Replacing Static Host Confirmation Message with JavaScript Confirmation Pop-up Box

Replacing static host confirmation messages (such as: "Are you sure? (Y/N)__") with JavaScript confirmation pop-up box provides a more Web oriented look and feel to the application functionality.

▶ **To replace the confirmation message with a JavaScript confirmation pop-up box**

1   Within the generated page's Java/C#/VB code create a method ("myConfirmMethod") that executes host logic that submits the data entered in the JSP page to the current host screen and performs the confirmation in the screen that follows.

2   Add the following function:

JSP:

```
function confirmSubmit(){
   if (confirm('Are you sure?')){
      gx_postBack('MyConfirmMethod'); // Execute a server side method
   }
   else{
      // do nothing
   }
}
```

.NET

Add the following button:

```
<input runat="server" type="button" id="confirmButton" value="GO" ↵
onserverclick="MyConfirmMethod" style="display:none" />
```

Then add the following JavaScript:

```
function confirmSubmit(){
   if (confirm('Are you sure?')){
     gx_getElement("confirmButton").click();
   }
   else{
      // do nothing
   }
}
```

3   Execute this function in any JavaScript event on the page.

For example:

```
<input type="button" value="Save" onClick="confirmSubmit();" >
```

You can even override the PF-key used to submit the data to perform this confirmation. For further details about setting keyboard mappings using JavaScript refer to **Mapping Keyboard Keys to User Actions in Individual Pages**.

Opening an Independent Pop-up Box that doesn't have a Corresponding Host Screen

ApplinX web applications handle host application modal windows by opening the modal window's corresponding JSP/ASPX page in a pop-up window. The main window is automatically locked, and any attempt by the user to interact with it just results in the pop-up window receiving the focus. In order to create a popup window (JSP/ASPX) that behaves the same way but has no corresponding host screen, carry out the following steps:

▶ **JSP**

1   Add a new Java class to the contexts package that extends the `GXBasicContext` class (in this example we'll refer to it as ExamplePopUp.java):

2   Create a new JSP file in your project. The initial JSP code looks like this:

```
<%@ page extends="com.sabratec.j2ee.framework.web.GXJspServlet"%>
<%@ taglib uri="com/sabratec/j2ee/framework/tags" prefix="gx"%>
<%@ page contentType="text/html; charset=utf-8"%>
<gx:page gx_context="contexts.ExamplePopUp">
<html>
 <head>
  <title>Insert title here</title>
 </head>
 <body >
  <gx:form>
  ...
  </gx:form>
 </body >
</html>
</gx:page>
```

> 📄 **Note:** Set the `gx_context` attribute of the `gx:page` tag to the newly created Java class.

3   Edit your code class: Override `ExamplePopUp.java onLoad` event method:

```
public void gx_onLoad(){
 try {
  gx_attach();

 }catch (GXGeneralException e) {
  gx_handleSessionError(e);
 }
  super.gx_onLoad();
 }
```

> **Note:** The `gx_attach` call is required, even when no interaction is made with the host application. It allows the framework to treat this window like any other host modal window. This means that the window will always stay in focus and the parent window will be "locked" until the window is closed.

4　There are a number of options to close the window:

- The Browser's Close button (the X button). The window closes and the main screen is refreshed.

- Calling the JavaScript gx_window.close() function. The window closes and the main screen is refreshed.

- To avoid the refresh effect you can handle the window closing with a server side method that will allow you to cancel the refresh in addition to enabling setting values in inputs or labels on the main screen and performing host related operations such as, sending host keys or executing path procedures. In order to do this add a server side method that handles the window closing functionality:

```
public void doClose(){
 // Cancel Parent page refresh upon closing the window
 getGXWindow().getOpenerWindow().cancelRefresh();

 // Set value of either an input or a label on the parent screen
 //getGXWindow().getOpenerWindow().setField("<input or label ID>", "<value>");

 // Close the window
 getGXWindow().close();
}
```

5　In the page from which you wish to invoke the popup, add the following HTML code:

```
<input type="button" value="openWin"
onclick="gx_window.open('<Path>ExamplePopUp.jsp','myPopUp','')"/>
```

Regarding the path, if ExamplePopUp.jsp is not within the root directory of the web application, an absolute path should be added (for example: "myDirectory/ExaplePopUp.jsp").

▶ .NET

1　Add a new Web Form (including a code class) to the web application (call it ExamplePopUp.aspx) . Open the code class .NET created and specify that the new class should inherit from `GXBasicWebFrom` rather than from `System.Web.UI.Page`..

2　Edit your code class: Override `ExamplePopUp.java onLoad` event method:

```
protected void Page_Load(object sender, EventArgs e)
 {
        gx_attach();
} ↵
```

> **Note:** The `gx_attach` call is required, even when no interaction is made with the host application. It allows the framework to treat this window like any other host modal window. This means that the window will always stay in focus and the parent window will be "locked" until the window is closed.

3   There are a number of options to close the window:

- The Browser's Close button (the X button). The window closes and the main screen is refreshed.

- Calling the JavaScript gx_window.close() function. The window closes and the main screen is refreshed.

- To avoid the refresh effect you can handle the window closing with a server side method that will allow you to cancel the refresh in addition to enabling setting values in inputs or labels on the main screen and performing host related operations such as, sending host keys or executing path procedures. In order to do this add a server side method that handles the window closing functionality:

```
public void doClose(object sender, EventArgs e)
{
   // Cancel Parent page refresh upon closing the window
   gx_window.getOpenerWindow().cancelRefresh();
   // Set value of either an input or a label on the parent screen
   //getGXWindow().getOpenerWindow().setField("<input or label ID>", "<value>");

   // Close the window
   gx_window.close();
}
```

4   In the page from which you wish to invoke the popup, add the following HTML code:

```
<input type="button" value="Open Win" onclick=
"gx_window.open('<path>/ExamplePopUp.aspx','myPopUp','')" />
```

Regarding the path, if ExamplePopUp.aspx is not within the root directory of the web application, an absolute path should be added (for example: "myDirectory/ExaplePopUp.aspx").

16 **Working with Tables**

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)

▶

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.

17   **Transferring Files (FTP)**

In ApplinX framework, it is possible to transfer files from the client to the host or from the host to the client, using the FTP dialog screens. The HTML emulation contains a link in the footer that opens an FTP Web dialog box.

FTP Configuration

To upload/download files using the FTP option, you need to configure a number of parameters in the Framework Configuration Editor.

▶ **FTP Configuration:**

1   Open a new browser and run your Web application.

2   Click on the Framework Configuration link. The Configuration Editor will be displayed.

3   Expand the **FTP** node.

4   Select the **Host type**.

5   Enter the **Host address**.

6   Click **Save** to save your changes.

7   Click **Close** to return to the Web application.

8   When working with an application which is not an HTML emulation, add an element such as a link or a button which when clicked will call the gx_openFtpDialog() JavaScript function (in an HTML emulation this is built-in).

Refer to **Using FTP to Upload Files** and **Using FTP to Download Files**.

Opening the File Transfer Dialog Box

To work with an application which is not an HTML emulation, add an element, such as a link or a button, to your master page (template.jsp/template.master.cs) or to any generated page. Place it in a suitable location. Set the link to call `gx_openFtpDialog();`.

Example:

```
<input type="button" id="FTPtButton" value="FTP" onClick="gx_openFtpDialog();" />
```

Refer to the API:

- `gx_openFtpDialog`

Using FTP to Upload Files

▶ **To upload files:**

1   Configure the FTP parameters as detailed in **FTP Configuration**.

2   Open the ApplinX HTML Emulation and run your Web application.

3   Click on the upload image to display the Upload dialog box.

4   Click on the **Upload** button. The File Upload screen will be displayed.

5   Enter the **User name** and **Password** (mandatory).

6   Enter the name of the file you would like to upload (Remote file). (Mandatory).

7   Click **Browse…** to enter the location and file name of the uploaded file.

8   Fill in the host property fields (these fields are optional and differ according to the configured host):

AS/400 Hosts:

1. Select the data representation type: ASCII, EBCDIC, IMAGE, DBCS_EBCDIC, EBCS_EBCDIC or CCSID.

2. Select the structure of the data that is to be transferred: File or Record.

3. Select the mode Stream or Block to determine whether records are transmitted record-by-record or as a continuous stream of bytes.

Mainframe Hosts:

1. Select the data representation type: ASCII, EBCDIC, IMAGE, UNICODE 2 B or UNICODE 2 L.

2. In the **Record format** field, specify the type of records in the data set: Fixed, Variable or Undefined.

3. In the **LRECL** field, specify the logical record length (in bytes).

4. In the **Block size** field specify the physical length of the data (in bytes).

5. In the **Primary** field specify the number of tracks or blocks initially allocated to the data set.

6. In the **Secondary** field specify the number of tracks or blocks if the primary allocation is exceeded.

9   It is possible to enter a user defined command in the **Command** field.

10 Click **Upload**. The upload process may take some time and is dependant on the size of the file and the connection.

11 A message will be displayed indicating that the upload was successfully completed. If there is a failure when uploading the file, an error message will appear.

Using FTP to Download Files

▶ **To download:**

1   Configure the FTP parameters as detailed in **FTP Configuration**.

2   Open the ApplinX HTML Emulation and run your Web application.

3   Click on the download image to display the Download dialog box.

4   Click on the **Download** button. The File Download screen will be displayed.

5   Enter the **User name** and **Password** (mandatory).

6   Enter the path and file name of the source file that is to be downloaded (mandatory).

7   Fill in the host property fields (these fields are optional and differ according to the configured host):

AS/400 Hosts:

1. Select the data representation type: ASCII, EBCDIC, IMAGE, DBCS_EBCDIC, EBCS_EB-CDIC or CCSID.

2. Select the structure of the data that is to be transferred: File or Record.

3. Select the mode Stream or Block to determine whether records are transmitted record-by-record or as a continuous stream of bytes.

Mainframe Hosts:

1. Select the data representation type: ASCII, EBCDIC, IMAGE, UNICODE 2 B or UNICODE 2 L.

2. In the **Record format** field, specify the type of records in the data set: Fixed, Variable or Undefined.

3. In the **LRECL** field, specify the logical record length (in bytes).

4. In the **Block size** field specify the physical length of the data (in bytes).

5. In the **Primary** field specify the number of tracks or blocks initially allocated to the data set.

6. In the **Secondary** field specify the number of tracks or blocks if the primary allocation is exceeded.

8   It is possible to enter a user defined command in the **Command** field.

9   When the file is a text file, it is possible to determine whether you would like to convert the downloaded file to Windows text mode, DOS text mode or UNIX text mode.

10   Click **Download**. The Windows File Download window will be displayed enabling you to save or open the file. If there is a failure when downloading the file, an error message will appear.

18 Printlet Servlet Redirector for AppinX

AppinX supports printer sessions on AS/400 and mainframe hosts. AppinX connects to the host, retrieves the print buffers and analyzes them. The host handles the printer's queue and the connection of printer sessions to display sessions. AppinX connects to the host as a printer session to receive the print buffers and allows you to work with them.

One of the ways you can print with AppinX is using the default behavior of an emulator, where AppinX sends all print jobs to the client's machine by means of an applet that runs on the client's browser. It is a signed applet, as it is Java code and it invokes the client machine's print dialog, which requires permissions that exceed regular applet permissions. Once the applet is signed it can only address the server where the applet originated, and therefore requires that the Web server and AppinX server are on the same machine. When it is not possible to have the Web server and AppinX server on the same machine, it is necessary to use the Printlet Servlet Redirector for AppinX.

The Printlet Servlet Redirector enables connection of the printlet to the Web server machine through the public HTTP/S port. The printlet server URL is defined as the URL of the Redirector Servlet instead of the AppinX server URL. In this way the Servlet Redirector redirects the data coming from the Printlet to AppinX server and vice versa.

In the current implementation, this solution does not support architectures with load balancing or clustering, as all communication of the printlet must go through the same Web session of the same Web server machine.

> **Note:** This feature is currently available for the JSP framework only.

▶ **To install and activate the Printlet Servlet Redirector for AppinX**

1   Activate the Redirector:

1. Edit the *web.xml* file located under your web project (*<web_dir>\WEB-INF*) as follows: Copy the commented section labeled "Listeners" from *\new jsp\new_application\WEB-INF\web.xml* to your *web.xml* and uncomment the listener tag.

```
<listener>
<listener-class>
com.sabratec.util.net.redirector.http.servlet.GXRedirectorServletSessionListener
</listener-class>
</listener>
```

2. Add the following tags:

```
<context-param>
<param-name>GXServerURL</param-name>
<param-value>applinx://localhost:2323</param-value>
<description>ApplinX server URL</description>
</context-param> ↵
```

Change the server URL to point at the machine where ApplinX is running.

3. Edit the *run_printlet.jsp* file which is in your Web application root directory: Change the param serverURL to `http://<web_server_name>:8080/<web_app>/z_redirector.jsp`

4. Restart the Web server.

2   Make sure that in your architecture, that the same IP address (or machine name) of the Web server, where the servlet redirector is enabled, is used for the following:

1. Calling the page *run_printlet.jsp* in the browser.

2. `web_server_name` set in the printlet parameters (in *run_printlet.jsp*).

19 Framework Management

Upgrading a JSP Web Application

> **Note:** Close all instances of Eclipse before commencing with the upgrade process. After the upgrade process has been completed, reopen Eclipse and refresh the project.

▶ **To upgrade an existing web application:**

1 In the ApplinX Explorer, right-click on the relevant application and select **Web Application Manager…** .The *Web Application Manager Wizard* is displayed.

2 Select **Upgrade an existing Web application**. Click **Next**. The *Web Application Folder* screen is displayed.

3 Locate and select the folder of the Web application to be upgraded, or the Eclipse project where the application is located. It is highly recommended to backup the Web application before upgrading. Click **Next**.

4 The *Wizard Summary* screen is displayed.

5 Click **Finish**. The Console area indicates whether the process succeeded or failed.

Deploying an ApplinX Web Application (JSP)

▶ **To deploy the Web application:**

1 Right-click on the relevant application and select **Deployment Manager for J2EE...**. The *Deployment Manager wizard* is displayed.
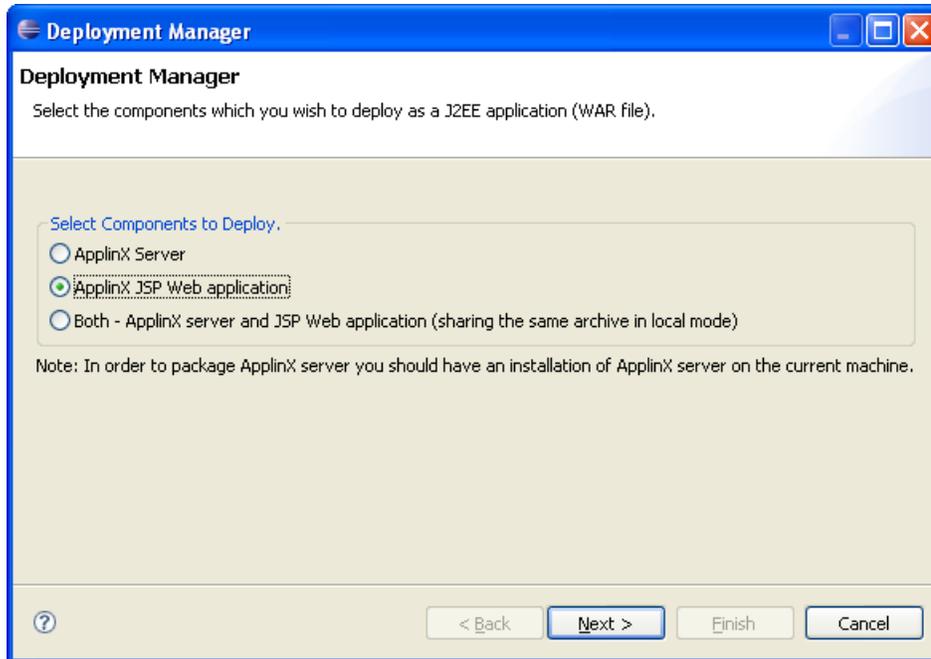
2   Ensure that the **JSP Web application** option is selected.

3   Click Next. The Select ApplinX Application screen is displayed.
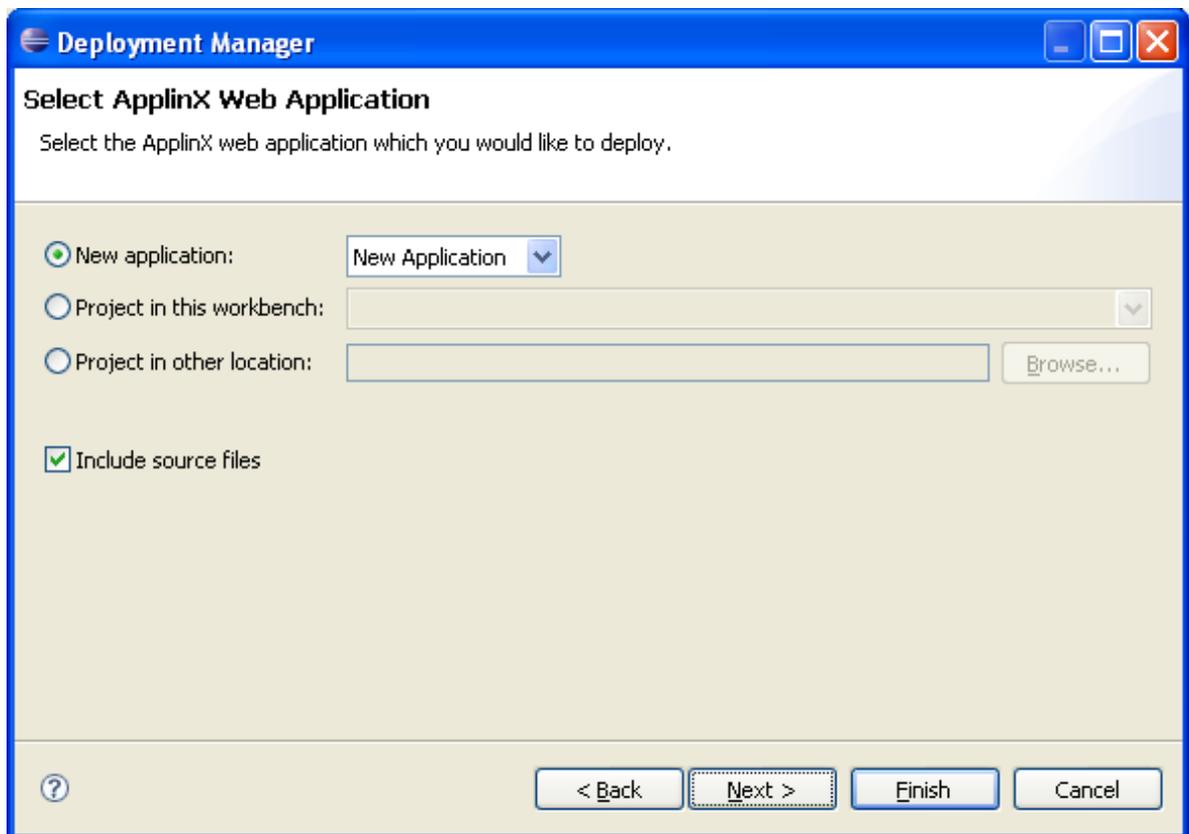
4   Select whether to deploy a new application or HTML emulation, or a project from within the workbench, or a project from a different location.

5   Select **Include source files** to include the Java files as well as the compiled classes. Click **Next**. The *Select Package Parameters* screen is displayed.
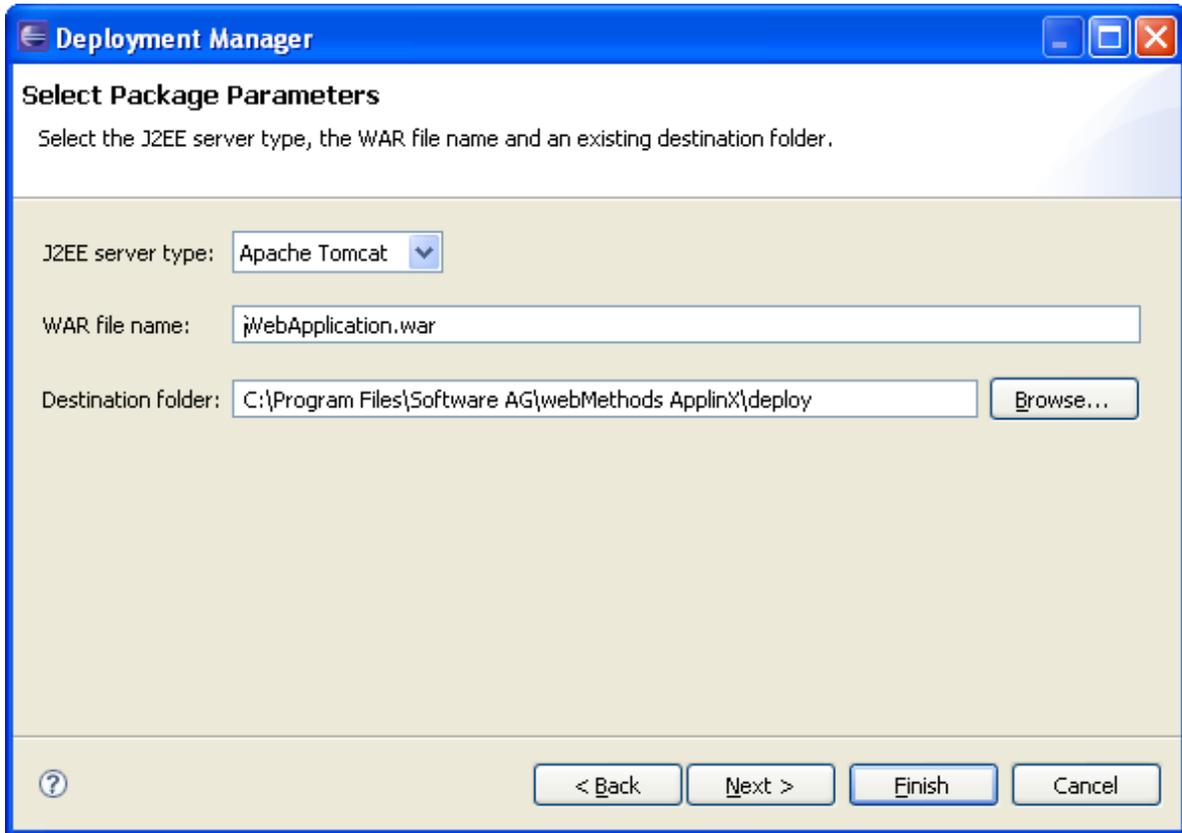
6   Select the server type, the WAR file name and the destination folder where the output file will be placed.

7   Click Next. The Select Target ApplinX Server screen is displayed.
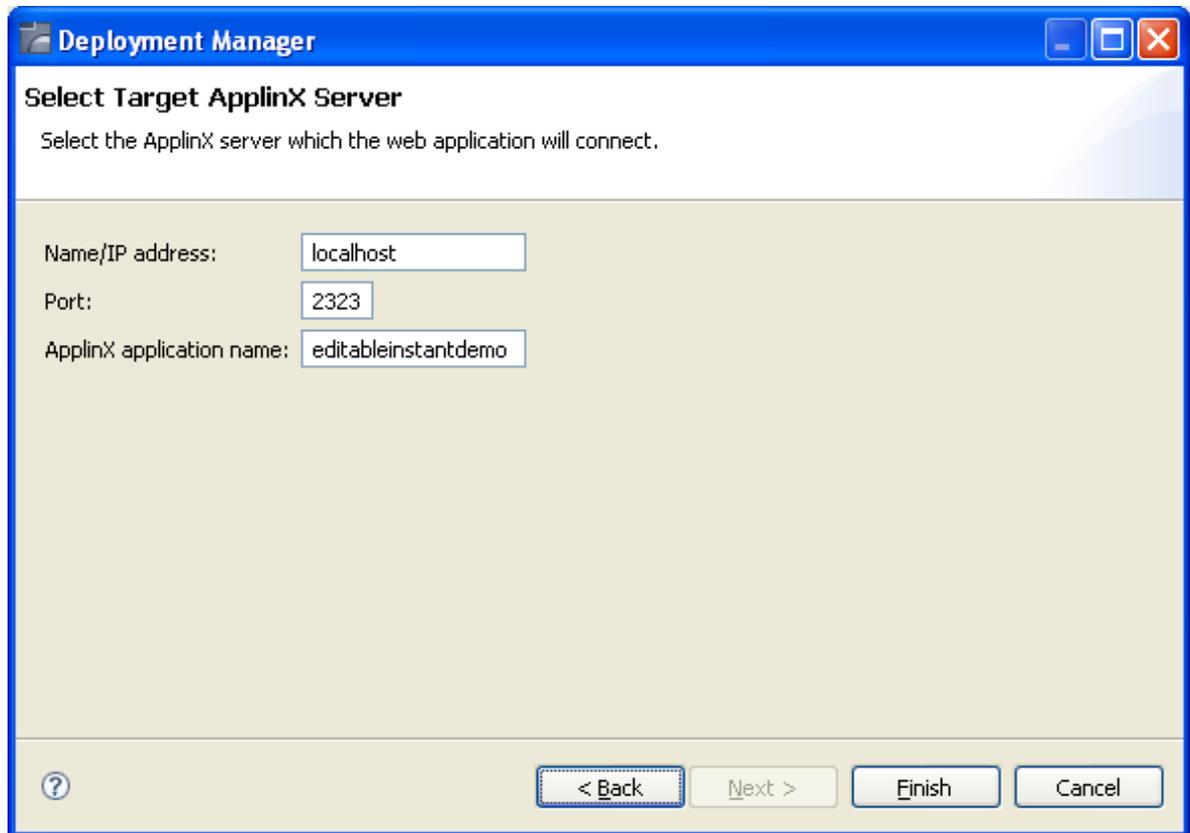
8   In this screen, Enter the ApplinX server and the port with which the Web application will connect. Also enter the ApplinX application name.

9   Click **Finish**.

10 In the console area, it is possible to see whether the process succeeded or failed. The WAR file created should be placed in your Web server directory.

▶ **Consider changing the following for production use:**

1   Change the framework logging according to your needs either by accessing the "Framework Configuration Editor" or in config/gx_logConfig.xml file. For example:

```
<category additivity="false" name="com.sabratec">
        <level value=" ERROR"/> <!-- Sets the log to Errors only -->
         <appender-ref ref="FRAMEWORK_LOG"/>
</category>
```

Possible values are:

- INFO - Normal
- WARN - Warnings

- ERROR - Errors only
- DEBUG – Debug

2   Disable the Performance monitor either by accessing the "Framework Configuration Editor" or in config/gx_appConfig.xml file set the WritePerformanceLog to false (this is the default configuration).

3   Disable the Javascript log using the config/gx_clientConfig.xml by setting LogLevel to 0 and ShowLogConsole to false (this is the default configuration).

4   Remove "Framework Configuration Editor" link from index page: delete the folder z_admin and remove the link to "configuration editor" from the index page.

Upgrading a .NET Web Application

▶ **To upgrade an existing Web application:**

1   In the ApplinX Explorer, right-click on the relevant application and select **Web Application Manager…** .The *Web Application Manager Wizard* is displayed.

2   Select **Upgrade an existing Web application**. Click **Next**. The *Web Application Folder* screen is displayed.

3   Locate and select the folder of the Web application to be upgraded. It is highly recommended to backup the Web application before upgrading. Click **Next**.

4   The *Wizard Summary* screen is displayed.

5   Click **Finish**. The Console area indicates whether the process succeeded or failed.

Deploying an ApplinX Web Application (.NET)

When deploying an ApplinX application that also consists of a Web application, for example when moving from the development environment to the production environment, the deployment consists of two parts: deploying the ApplinX application and entities and deploying the Web application.

> **Note:** ApplinX framework supports .NET clustering in a Web farm environment.

▶ **To deploy the Web application:**

1   Copy the entire Web application into a temporary working folder.

2   In the *config*/*gx_logConfig.xml* file edit the Logger settings according to your needs (enable/disable the log, target path and log level). For example:

```
<logger name="com.sabratec">
        <level value="INFO"/> <!"sets the log to normal mode
        <appender-ref ref="FRAMEWORK_LOG"/>
</logger>
```

When you do not require the performance log, ensure that the last two category nodes in the XML file are commented.

3   Change the definition of `designMode` in *config*/*gx_appConfig.xml* to "false".

4   Change the definition of the serverURL and applicationName in *config*/*gx_appConfig.xml* to the target server.

5   Copy the updated folder to the relevant Web server.

6   Map this folder as a virtual directory under the server.

Disconnecting the Host Session Correctly

When running the Web application in a browser there are times (such as when closing a tab, terminating a browser process etc.) that the browser close function is not triggered. Therefore, one cannot assume that when clicking the "X" close window button that the session will be disconnected. To assure disconnecting the session, it is recommended to set a timeout (in the ApplinX Designer>Host configuration tab, Connection timeout field) that will automatically disconnect the session after the predefined amount of time passes.

It is highly recommended to train the user to logoff from the application using the Logoff link. Optionally, In both .NET and JSP, it is possible to configure an event class which notifies when a session ends.

▶ **To customize the logoff process from the code:**

■   JSP: See `HttpSessionListener`

.NET: See usage of `global.asax` in ASP.NET documentation

III Troubleshooting the Framework

20 **Performance Monitoring**

Performance monitoring provides the ability to trace the performance of all sessions/single sessions, and save them to a CSV (Excel) file.

This feature monitors the process of an instant/generated page, from pressing a host key until the result page is fully loaded. The monitoring includes browser performance, Web server performance and ApplinX server/host performance.

The monitoring includes:

- The entire wait time the user waits between pressing an action key, until receiving the HTML result of the target screen - Total Request.

- The entire time that the Web server processed the user request (including ApplinX server & the host response times) - Total Server Side.

- The send key response time - SendKeys.

- The HTML page loading time - Client Load .

- Important ApplinX actions such as attaching to ApplinX server, creating the instant page (Generate Instant), updating controls from the host screen, updating tables (when tables exist), detaching from ApplinX server.

21 **JavaScript Logger Engine**

The JavaScript logger engine is used to log JavaScript errors (primarily) as well as to debug specific modules of the JavaScript engine. The engine logs JavaScript errors which occur to users of the Web application. These errors are logged to the framework JavaScript log, with the following user information: Session ID, IP, browser details.

Each line in the log displays a timestamp, module name, log level, and the message.

Logging JavaScript Errors

The JavaScript logger engine automatically logs JavaScript errors to a log text file on the server (*logs/javascript_log.txt*).

> **Note:** Applications created in ApplinX versions prior to 5.2.4 this file should be replaced/edited (if changes were made) with the *config/gx_logConfig.xml file* from the new application/config directory of the relevant framework type jsp/c#/vb.

Debugging JavaScript Modules

The JavaScript Logger Engine can be used by ApplinX developers to debug specific modules of the JavaScript engine (The JavaScript engine core, Emulation, modal windows, type ahead, Natural data transfer), by displaying the information in the log console (or writing to the server log when the log console is not shown).

To configure the JavaScript log console, configure parameters in the *config/gx_clientConfig.xml* file as follows:

```
<engineConfig
logLevel="3" <- 3-activate debug mode (0-Error , 1-Warnings , 2-Massage , 3-Debug)
debugModules=:z_engine.js,z_emulator.js,z_window.js,z_typeAhead,z_ndt.js" <- which ↵
modules to debug
showLogConsole="true" <- show/hide the log console
/>
```

> **Note:** The log text file also logs any error/debug (when active) messages when the browser is refreshed, when the system is disconnected, when clicking on the **Write to server log** button in the log console (if the log console is displayed) and when the log console's size reaches 5k (to avoid reducing the speed of the browser).

Example:

Whenever myFunc is executed the Boolean variable "myBool" is evaluated. If the value received is "false", a message is logged in the debug log stating that there is a problem.

```
function myFunc(){
      var myBool = false;
      doSomthing(myBool);
      if (!myBool){
       GXLog.debug("myModule","Something went terribly wrong!!");
      }
      return myBool;
}
```

Ensure to set the *config/gx_clientConfig.xml* as detailed above.

Refer to the API:

- GXLog.debug(moduleName, message)

- GXLog.warning(moduleName, message)

- GXLog.error(moduleName, message)

22 Debugging/Analyzing the Web Application's Code

It is possible to use the ApplinX framework log to analyze and debug the code in the Web application. This is implemented by making changes in the *config/gx_logConfig.xml* file.

▶ **To define the messages that will be displayed in the framework log:**

1   Open *config/gx_logConfig.xml*.

2   Uncomment the following section:

   JSP:

```
<!--category additivity="false" name="<Category-Name>">
 <level value="info"/>
 <appender-ref ref="FRAMEWORK_LOG"/>
</category
-->
```

   .NET

```
<!-- logger additivity="false" name="<Category-Name>">
 <level value="info"/>
 <appender-ref ref="FRAMEWORK_LOG"/>
</logger
-->
```

3   Change the level tag value to either: info, error, warning, or debug.

4   Change the <Category-Name> to a meaningful value. This will make it easier for you to find log entries at a later time.

5   Restart your web server, in order for the changes to take effect.

6   To write to the log file from anywhere in your code, make sure that the call looks similar to the following (depending on the log level):

```
GXLog.info("Category-Name","<Message>");
GXLog.error("Category-Name","<Message>");
GXLog.warning("Category-Name","<Message>");
GXLog.debug("Category-Name","<Message>");
```

IV ApplinX Development API References

23 Web Application Configuration Parameters

ApplinX provides a Web based configuration editor where you can configure framework parameters. These parameters are saved in the *config/gx_appconfig.xml* file. A description of each parameter appears in the Javadoc on the right side of the window (click on a parameter/node to display the relevant Javadoc).

> **Note:** These parameters can be manually configured in the *config/gx_appconfig.xml* file.

This chapter covers the following topics:

Session Parameters

Parameter	Description
Server URL	The ApplinX server URL. By default it is applinx://localhost:2323. For secured SSL socket: applinxs://localhost:23443.
Application name	The name of the ApplinX application.
Session ID	The Session ID in the web server. Can be set according to the connecting IP address (IPv4 and IPv6 address formats are supported) or by an allocated session ID. It is also possible to set the Web Session ID from the code (in the GXBasicContext (JSP)/GXBasicWebForm (.NET) file), overriding the option selected here.
Password	The password required to access a specific session on the ApplinX server. Can be set according to the connecting IP address or by an allocated session ID. It is also possible to set the password from the code (in the GXBasicContext (JSP)/GXBasicWebForm (.NET) file), overriding the option selected here.
Additional	
Description	The description of the session in ApplinX server. This description appears in the list of sessions in ApplinX Administrator. The information entered can be dynamic and may include tokens such as: $(SESSION_ID): Represents the Web server session ID. $(IP): Represents the user IP address. $(BY_APPLINX): Represents the ApplinX session ID.
Connection Pool	The connection pool name used to get a session. When a connection pool name is stated, the framework will connect to the connection pool instead of using a new session.
Host user name	The host user name. Relevant for SSH and Natural-Unix protocols only.
Host password	The host password. Relevant for SSH and Natural-Unix protocols only.
Show Intermediate screens	Determine whether to show intermediate host screens. Relevant for Natural Set Control N command.

Instant Parameters

Parameter	Description
Font size	It is possible to change the default font size to a different size (in pixels). Since the change affects the rendering process performed by the framework on the Web server, this setting is not part of the CSS style sheet file (unlike other font settings), but instead it is included in the server-side instant configuration parameters. Note: Changing the font sizes for generated Web pages is done in the CSS style sheet node. The possible values include the different pixel possibilities as well as "Dynamic by Resolution". The HTML emulation recognizes user resolution and dynamically changes the css name according to the matching resolution. The resolution can be either 640x480, 800x600, 1024x768 or 1280x1024 pixels. The resolution may also depend on the host resolution (80 or 132 characters). The best-fit functionality is configured by default in the emulation template in all frameworks. It is used by the renderer to calculate the top/left according to each tag's position.
Font family	Sets the font family to be used for instant rendering. It is used for proportion calculation to best fit specific fonts for Japanese & Arabic applications. For Japanese the best fitted fixed fonts are: "MS Mincho" or "MS Gothic" For Arabic the best fitted fixed font is: "Courier". For other languages the best fitted font is "Courier New", which is by default declared in the CSS files. Defining the font here will override the CSS definition.
Row spacing	Sets the row height to be used for instant rendering. This height is calculated relatively to the row's font size. The default row height is set to 53% (153% of the font size). For example: When the font size is set to 13px, and using the default row height value of 53%, the row height will be:13*1.53 =19.89= ~20px.
Color mode	Determines whether to display all the colors as they are displayed in the host (background & foreground), display only the background colors or not display any colors at all.
Host keys	The type of the rendered host keys.
Render tags from right	Determines that the HTML tags will be positioned from right to left. The usage is in Hebrew/Arabic applications when using non-fixed font, to ensure alignment to the right. By default the tags are positioned from left to right.
Render emulation behavior	Determines whether to render emulation attributes in text fields. Emulation attributes are special ApplinX attributes that start with "gx_" and are added to the rendered text field tag. These attributes provide additional information to the ApplinX JavaScript engine, which changes the behavior of the text fields. The attributes are: ■ Automatic skipping (gx_autoSkip) - will cause the text field to skip to the neighboring text field, once it is filled with content. ■ Data type (gx_dt) - The data type of the field. See GXBaseObjectConstants for possible data types. ■ Right adjustment (gx_ra) - AS/400 right adjust. ■ Automatic ENTER (gx_au) - AS/400 automatic enter.

Parameter	Description
Enable displaying tables	Determines whether the relevant transformation (such as displaying tables, or displaying the graphical window frame) is enabled.
Enable displaying graphical window frames	Determines whether the relevant transformation (such as displaying tables, or displaying the graphical window frame) is enabled.
Define render area	Used to define that a specific area of the host screen will be rendered as HTML in Instant HTML pages by specifying the area's boundaries and in this way hide certain portions of the screen. By default, the entire screen is configured to be rendered to the HTML output. When setting the area to be rendered, tags positioned outside the rendered area will not be available for query execution on the screen tag model. The rendered area also affects the rendered top/left attributes of the tag. The top/left tag will be calculated relatively to the starting position of the rendered area.

General

Parameter	Description
Use Folders – virtual directory	Determines whether the Web application folder structure should be the same folder structure as configured in the ApplinX repository. Relevant for screens only. This is recommended when the Web application contains a large number of designed Web pages. Virtual directory: Determines the virtual directory used for the project. Should be without any slashes. Needs to be set only if UseFolders is true. For a site (with domain) the value should be empty string. For example: ■ If the web application URL is <http://localhost:8080/myApp/index.jsp>, the virtual directory is "myApp". ■ Also for a web site http://myWebsite/myApp/index.jsp <http://mywebsite/myApp/index.jsp> , the virtual directory is "myApp". ■ If for example I have two folders <http://localhost:8080/myApp/mysubApp/index.jsp>, the virtual directory is "myApp/mySubApp".
Use screen locker	The purpose of a screen locker is to indicate to the user by means of a message, that the application is processing his request, and blocks him from interfering with the current process by repressing a button/link or keyboard PF/ENTER. To activate the screen locker first select the Use screen locker check box and then access the template/screenLocker.htm file and use the width/height percentages to control the location of the message. Replace the text "Please wait" with an alternative text/image as required. Refer to the documentation for further details.

Parameter	Description
Alternate row colors	Determines whether to use alternate colors for every other row in a table. Use AlternateCssName to control the alternate css name.
Prevent page refresh effect	This feature prevents the displayed page from flickering every time the page is submitted. This affect is prevented by using two separate frames, while only one is visible at each stage. These frames continuously switch between being active and passive each time a page is submitted. The active frame is displayed, and every time a page is submitted, the frame becomes the passive frame, in this way preventing the flickering affect. When the modal window feature is enabled in the Framework Configuration Editor or the 'Enable Natural-Data-transfer support' is enabled in the host, this feature is automatically enabled even if this parameter is not selected.
Perform background check for host screen changes	Determines whether to continuously check in the background for changes in the host screen. If the host screen has changed, the browser is refreshed and a JavaScript event is triggered. As the server is checked continuously, more server resources are required for this. By default, the server is checked every 3, 6, 12 and then every 24 seconds. These intervals can be customized using user exits (refer to the ApplinX documentation for further details).

Logoff

Parameter	Description
Disconnect host session when browser is closed	Determines whether to log off from the host session when the browser is closed. It is recommended to define a session timeout in the ApplinX Designer (Application Properties dialog box>Host tab, Non-activity timeout) to ensure logging off from a session.
Display the following confirmation message	The confirmation message which is displayed by default is displayed in this field. You can change the message as you see fit.
Prompt user before session time-out	Set to true to display a message indicating to the user that the session is about to be disconnected. This message will be displayed towards the end of the Non-activity timeout period (defined in the Application Properties Host tab). The user will be able to select to resume or quit the session. When not selecting either of these options, the user will be logged off automatically. It is possible to customize the relevant prompt page (template/logoffPrompt.htm).
Termination path	Defines a termination path to use when the user logs off by either closing the browser, or clicking the logoff link (activates logoff.jsp/aspx).

Generated Pages

Parameter	Description
Reflect host protected - Dynamically disable CSS	Determines whether the protected fields in the host will be read only in the Web application (when gx_fillForm is called). Dynamically disable CSS: The css class that is used to give a different look to a read-only input field, which becomes read-only according to the value of the ReflectHostProtected parameter.
Reflect foreground colors	Determines whether the field foreground color in the Web application will be the same as the field foreground color in the host. The relevant css classes are in css/styles_generated.css file.
Reflect background colors	Determines whether the field background color in the Web application will be the same as the field background color in the host. The relevant css classes are in css/styles_generated.css file. /css_colors folder in the emulation template.
Reflect maximum field length	Determines whether the fields' maximum length attribute in the host will be the maximum length attribute in the input fields in the Web application. This saves you the work of manually adding this attribute to each input field when designing the page.
Reflect emulation behavior	Determines whether to add emulation attributes in text fields. Emulation attributes are special ApplinX attributes that start with "gx_" and are added to the rendered text field tag. These attributes provide additional information to the ApplinX JavaScript engine, which changes the behavior of the text fields. The attributes are: ■ gx_autoSkip - will cause the text field to skip to the neighboring text field, once it is filled with content. ■ gx_dt - The data type of the field. See GXBaseObjectConstants for possible data types. ■ gx_ra - AS/400 right adjust ■ gx_au - AS/400 automatic enter

Window

Parameter	Description
Enable modal windows - opened window top, opened window left, opened window attributes, Host key to send when the window is closed	Determines whether to activate ApplinX pop-up manager and transform recognized host windows to Web pop-up windows.
Set window position to center	Determines that the popup window is opened in the center of the screen.

Parameter	Description
Set window specific position	Determines that the popup window is opened in a specific position.
Opened window attributes	Determines the opened window's attributes when using the method gx_window.open
Host key to send when the window is closed	Determines the key to send to the host when the modal window is closed

Emulation

Parameter	Description
Support type ahead	Determines whether to enable the user to type continuously, without waiting for the browser to display the next page. This feature uses an ActiveX component (works in IE only). This feature does not work when a modal window is open, or during the process of opening/closing a pop-up window.
Support Dup and FieldMark host keys	Dup and FieldMark are special Mainframe characters. In Instant and generated pages, a user can send to the host a non printable character in specific input fields.
Use keyboard PF keys	Set when the PF keys in the browser are enabled. Default value: true.
Tab on input fields only	Determines whether the TAB key navigates between the input (unprotected) fields of the screen only, or also between additional buttons and hyperlinks in the Web page.
Automatic skip for all input fields	Determines whether or not the cursor will automatically move to the next input field once the current input field is filled with text (applies to all input fields).
Block illegal characters in host numberic fields	Determines whether it will be possible to type non-numeric characters in numeric input fields. The type of the field is a property of the host field which the input field is based on.
Select content when focus on input field	Determines whether when moving the cursor to the following field (using the arrows or TAB key), the field content will be selected and then overwritten or the text will not be selected and when typing, the text will be inserted in the field.
Show blinking caret	Displays a blinking caret in input fields
Show host blinking fields	Some hosts contain definitions determining that certain fields should blink. This feature enables these fields to blink in the browser.
Place caret at end of field	Determines whether the caret's position should be at the end of the input fields.
Use paste behavior as in terminal emulators	Determines whether the paste behavior of text will be as in terminal emulators, that once a field is filled, the remaining text is pasted in the

Parameter	Description
	following field or whether the text will be pasted in a single field. Only supported in Internet Explorer.

Natural upload/download

Parameters	Description
Use ActiveX for Natural data transfer (IE only)	Defining to use the ActiveX component, enables the work flow's behavior to be identical to that of Entire Connection. This is relevant for Internet Explorer only. **Note:** 1. ActiveX will work only if you have Microsoft .NET Framework version 3.5/4.0 installed on your machine. 2. The site using the ActiveX needs to be defined as a "trusted site". Under **Tools** > **Internet Options** > **Security**, click **Custom level** and under **ActiveX controls and plug-ins** > **Allow Scriptlets** check **Enable**.
Automatically start download	Determines whether to automatically start downloading a file, when the first download screen is displayed. When not selected, a message appears informing you that the host requires you to download a Natural file. If there is a record count from the download, it will be displayed on the page.
Download file in one go/Download file in blocks	According to the selected radio button, determines whether to download the file as a whole, or whether to download the file in blocks, appending each block to the existing file. This parameter is typically set when expecting to download large files and is only relevant when working with ActiveX and with a non Natural Unix protocol which supports Natural data transfer.
Default file extension	Enables defining the default file extension: TXT or CSV.

Log

Parameters	Description
File name	The log is written to this file.
Append to existing file	Selecting this check box determines whether when restarting the Web server, the log file will be overwritten or a new file will be created.
Log level	The contents of the log file are as detailed as this property defines, where every level includes the levels above it. For example, the Debug level also logs Normal, Warnings and Errors Only levels. Available values: "Normal", "Warnings", "Errors only" and "Debug" (by default Normal is selected).

Parameters	Description
Log history	Determines the number of backups saved before overwriting the old log files. For example: 10 means "save the last 10 log files, in addition to the current one, then start to overwrite".
Max. file size	Starts a new log file after the current file has been filled to the maximum file size.

Performance Monitor

Parameters	Description
Enable performance monitoring	Determines whether to create a performance log.
File name	The log is written to this file.
Write performance log per session ID	When selected, creates a performance log for each session.
Trace sessions	Determines whether to trace all sessions or only the session specified here.
Description	Short description means that the information is displayed in a tabular structure. Detailed description displays the information as a paragraph.

Macro

Parameters	Description
User name	The user macros file will be saved according to the user name, therefore it should be a unique ID of the Web user. The User name can be according to the IP Address, it can be cookie based, or set in the code. By default it is configured to IP Address, which is a good idea when the user's IP is constant. When selecting the cookie based option, ApplinX will assign a cookie to the end user, and will read the macro according to the cookie. Macros will be lost when the user deletes his cookies. To enable this option you should set SaveUserNameLocally to true. Select Other to use a session variable which was saved according to the login page user name, or to use the NT user name (relevant for .NET only) (Request.ServerVariables["AUTH_USER"])
Macros folder	Determines the folder on the server where the macros will be saved.
Encrypt macro file	Determines whether to encrypt the macro. Recommended to use when the macro includes user names and/or passwords. When changing this setting, existing macros will no longer function. The existing macros will be deleted from the macros folder when recording new macros.

Single Sign On

Parameters	Description
Enable Single SignOn	Determines whether to activate the single sign option.
SignOn screen recognition	Determines the method to use to recognize the sign on screen. According to the screen name. Automatically recognize the user name and password fields. Set from the code.
Screen name	The name of the SignOn screen, when recognizing the signOn screen according to the screen's name.
Define application field names	Returns the application field name which determines the user name field.
User/Password retrieving:	
Source	Determines the type of user/password retriever to use: ■ Based on information retrieved from an HTTP request. ■ Based on information retrieved from an HTTP session. ■ Set in the code.
User parameter name	
SignOn execution:	
Type	Determines what happens once the SignOn is recognized: ■ Using an action key (enter an action key). ■ Executing a path (enter a path name) . ■ Set in the code.
Action key	Enter the action key that is to be activated

FTP

Parameters	Description
Host type	Sets the type of the remote FTP host. Possible values are: Mainframe AS400 Other - for any other kind By Applinx - when the FTP host is the same as the host used in the current ApplinX session.
Host address	Sets the IP address of the remote FTP host.

CSS Classes

Parameters	Description
Instant only	
Window frame CSS class	The CSS class of window frame.
Host keys CSS class	The CSS class of the host keys
Table CSS class	The CSS class of the table tag.
Table Odd rows CSS	
Render intensified CSS class	Sets if to render intensified css class for intensified host fields.
Render application field CSS class	Sets if to render the application field css class.
Generated & Instant	
Table even rows CSS class	
Render intensified CSS class	Determines whether to render an intensified css class for intensified host fields.
Render application field CSS class	Determines whether to render a css class for application fields
Table even rows CSS class	
External CSS file parameter name	Defines an external css request parameter name. Used for portal integration. A portal application may provide to the framework as a query string a css URL as follows: http://?cssurl= ("cssurl" will be the value of the field in this case)

24   **Base Object**

Refer to the Base Object API. To view the Javadoc Development API, locate within your ApplinX installation the **api** folder (by default: \SoftwareAG_documentation\webMethods\ApplinX\9-6_ApplinX\api) or locate the API folder in the ApplinX>help folder (UNIX). The API includes documentation for all Base Object packages and classes, several utility classes and the ApplinX Load Balancer.

25 Server Side API (Java/.NET)

General

- User Exits
- GXIClientBaseObject
- ApplinX Abstract Web Classes – gx Building Blocks
- GXIScreenBasedForm
- Instant Component
- Host Keys Component
- Printer Control (.NET only)
- Useful JavaScript Functions

User Exits

An event is a procedure of the framework engine that informs you that a certain process occurred, and consequently allows you to capture this event and add your own code. The ApplinX events are called by the ApplinX framework building blocks.

- gx_preConnect
- gx_postConnect
- gx_preSendKeys
- gx_postSendKeys
- gx_screenSeqMismatch
- gx_changeNextForm
- gx_preSyncHostWithForm
- gx_preFillForm
- gx_postFillForm
- gx_downloadFile
- gx_getNdtDefaultDownloadFileName

gx_preConnect

Occurs before gx_connect, gx_attach.

JSP

Use `event.isNewSession()` to ascertain if it is before attach or connect.

Use `event.getSessionConfig()` to change the connection to the ApplinX server parameters.

Capture the event for the whole project from `GXBasicContext`, or from a certain page, in `YOUR_PAGE.java`, `OnInit` function.

For example, adding a device name:

```
}
public void gx_preConnect(GXIHostPageContext gx_context,GXPreConnectEvent event){
       if (event.isNewSession()){
event.getSessionConfig().addVariable(GXBaseObjectConstants.GX_VAR_DEVICE_NAME,"MY_DEVICE"));
}
}
```

.NET

Use `e.newSession` to ascertain if it is before attach or connect.

Use `e.sessionConfig` to change the connection to the ApplinX server parameters.

Capture the event for the whole project from `GXBasicWebForm`, or from a certain page, in `YOUR_PAGE.aspx.cs`, `OnInit` function.

For example, adding a device name:

```
protected override void OnInit(EventArgs e){
       base.OnInit(e);
       this.gx_preConnect += new GXPreConnectEventHandler(user_preConnect);
       ...
}
protected void user_preConnect(object sender,GXPreConnectEventArgs e){
            if (e.newSession){
e.sessionConfig.Variables = new GXVariable[] {new ↵
GXVariable(com.sabratec.applinx.baseobject.GXBaseObjectConstants.GX_VAR_DEVICE_NAME,"MY_DEVICE")};
   }
}
```

gx_postConnect

Occurs after `gx_connect`, `gx_attach`.

JSP

Use `event.isNewSession()` to ascertain if it is after attach or connect.

For example, skip messages screen, for all the project pages (in `GXBasicContext`):

```
public void gx_postConnect(GXIHostPageContext gx_context,GXPostConnectEvent event){

        if(gx_context.getGXSession().getScreen().getName().equals("MESSAGE_SCREEN")){

                gx_context.getGXSession().executePath("SKIP_MESSAGES_PATH");
}
}
```

.NET

Use `e.newSession` to ascertain if it is after attach or connect.

For example, skip messages screen, for all the project pages (in `GXBasicWebForm`):

```
protected override void OnInit(EventArgs e){
base.OnInit(e);
this.gx_postConnect += new GXPostConnectEventHandler(user_postConnect);
  ...
}
protected void user_postConnect(object sender,GXPostConnectEventArgs e){
      if (gx_session.getScreen().getName() == "MESSAGE_SCREEN"){
           gx_session.executePath("SKIP_MESSAGES_PATH");
}
}
```

gx_preSendKeys

Occurs before `gx_processHostKeyRequest(GXSendKeysRequest sendKeyRequest)`, which is activated from a browser PF key, if ENTER is pressed, or JavaScript command `gx_SubmitKey(key)`.

JSP

Use `event.getSendKeyRequest` to change the send key request to the ApplinX server.

For example:

```
public void gx_preSendKeys(GXIHostPageContext gx_context,GXPreSendKeyEvent event){
        if(event.getSendKeyRequest().getKeys().equals("[enter]")){
              event.getSendKeyRequest().setKeys("[pf3]");
}
}
```

.NET

Use `e.sendKeyRequest` to change the send key request to the ApplinX server.

For example:

```
protected override void OnInit(EventArgs e){
 base.OnInit(e);
 this.gx_preSendKeys += new GXPreSendKeyEventHandler(user_preSendKeys);
  ...
}
protected void user_preSendKeys(object sender,GXPreSendKeyEventArgs e){
      if (e.sendKeyRequest.getKeys() == "[enter]"){
            e.sendKeyRequest.setKeys("[pf3]");
}
}
```

gx_postSendKeys

Occurs after `gx_processHostKeyRequest(GXSendKeysRequest sendKeyRequest)`, which is activated from a browser PF key, if [ENTER] is pressed, or a `JavaScript command gx_SubmitKey(key)`.

JSP

```
    // used for performing actions after send keys
          public void gx_postSendKey(GXIHostPageContext ↵
gx_context,GXPostSendKeyEvent event)throws GXGeneralException{
          if (gx_context.getGXSession().getScreen().getName().equals( ↵
"DisplayMessage")){
          getGXSession().sendKeys("[enter]");
}
          }
```

.NET

```
protected override void OnInit(EventArgs e){
 base.OnInit(e);
 this.gx_postSendKeys += new GXPostSendKeyEventHandler(user_postSendKeys);
  ...
}

protected void user_postSendKeys(object sender,GXPostSendKeyEventArgs e){

      if (gx_session.getScreen().getName() == "DisplayMessage"){
            gx_session.sendKeys("[enter]");
}
}
```

gx_screenSeqMismatch

Occurs if the form sequence screen number is different from the `gx_session` sequence screen number.

JSP

Use `event.setSendToHost` to send the data to the host in any case.

For example:

```
public void gx_screenSeqMismatch(GXIHostPageContext ↵
gx_context,GXScreenSeqMismatchEvent event){
 if ↵
(gx_context.gx_getForm(gx_context.getGXSession().getScreen().getName()).equals(gx_context.getGXForm().getFormName())){

 event.setSendToHost(true);
 }
}
```

.NET

Use `e.sendToHost` to send the data to the host.

For example:

```
protected override void OnInit(EventArgs e){
base.OnInit(e);
this.gx_screenSeqMismatch += new ↵
GXIScreenSeqMismatchEventHandler(user_screenSeqMismatch);
}
protected void user_screenSeqMismatch(object sender,GXIScreenSeqMismatchEventArgs e){
 if (gx_getForm(gx_session.getScreen().getName()) == gx_form.FormName){
 e.sendToHost = true;
 }
}
```

gx_changeNextForm

Occurs before loading next page, by `gx_handleHostResponse`. Use `e.nextForm` to change the next page.

For example:

JSP

```
 public void gx_changeNextForm(GXIHostPageContext gx_context,GXChangeNextFormEvent ↵
event)throws GXGeneralException{
          if (gx_context.getGXSession().getScreen().getName.equals("SCREEN_A")){

              event.setNextForm("SCREEN_B.jsp");
}
      }
```

.NET

```
protected override void OnInit(EventArgs e){
this.gx_changeNextForm += new GXChangeNextFormEventHandler(user_changeNextForm);
}
protected void user_changeNextForm(object sender,GXChangeNextFormEventArgs e){
      if (gx_session.getScreen().getName == "SCREEN_A"){
            e.nextForm = "SCREEN_B.aspx";
}
}
```

gx_preSyncHostWithForm

Occurs before `gx_syncHostWithForm`, and its use is to add parameters to the map path that is executed by the framework. The map path in the framework is declared in:

JSP: `GXBasicContext : gx_appConfig.setMapPath("<APPLINX MAP PATH NAME>");`

.NET : `GXBasicWebForm : gx_appConfig.MapPath = "<APPLINX MAP PATH NAME>"`

For example:

JSP

```
 public void gx_preSyncHostWithForm(GXIHostPageContext ↵
gx_context,GXPreSyncHostWithFormEvent event)throws GXGeneralException{
event.getNavigateRequest().addVariable("CUSTOMER_ID", ↵
gx_context.getRequest().getparameter("CUSTOMER_ID"));
          }
```

.NET

```
 protected override void OnInit(EventArgs e){
this.gx_preSyncHostWithForm +=new ↵
GXPreSyncHostWithFormEventHandler(user_preSyncHostWithForm);
}
protected void user_preSyncHostWithForm(object sender,GXPreSyncHostWithFormEventArgs ↵
e){
e.navigateRequest.addVariable("CUSTOMER_ID", Request.QueryString["CUSTOMER_ID"];
}
```

gx_preFillForm

Occurs before `gx_fillForm()` or `gx_fillForm(GXIScreensCollection screen)`.

Use this user exit to fill in additional fields whose contents are not received directly from the current host screen.

For example:

JSP

```
     public void gx_preFillForm(GXIHostPageContext gx_context)throws ↵
GXGeneralException{
gx_context.getTagsAccesor().setTagContent("UserNameTitle",gx_context.getSession.getAttribute("UserName"));
          }
```

.NET

```
protected override void OnInit(EventArgs e){
this.gx_preFillForm +=new EventHandler(user_preFillForm);
}
protected void user_preFillForm(object sender,EventArgs e){
 UserNameTitle.Text = Session["UserName"];
}
```

gx_postFillForm

Occurs after `gx_fillForm()` or `gx_fillForm(GXIScreenCollection screens)`. Use this user exit to override the field data received from the host screen.

For example

JSP

```
public void gx_postFillForm(GXIHostPageContext gx_context)throws GXGeneralException{
gx_context.getTagsAccesor().setTagContent("CustomerID",gx_context.getSession.getAttribute("CustomerID"));
           }
```

.NET

```
protected override void OnInit(EventArgs e){
this.gx_postFillForm +=new EventHandler(user_postFillForm);
}
protected void user_postFillForm(object sender,EventArgs e){
    CustomerID.Text = Session["CustomerID"];
}
```

gx_downloadFile

Use this user exit to manipulate the content of a Natural downloaded file and change the content to formats such as RTF, CSV or to any other desired format.

For example

JSP

In `GXBasicContext`:

```
public void gx_downloadFile(String fileName, byte[] bytes) throws IOException {

     String fileContent = new String(bytes); // create a string from the bytes
     // manipulate here the fileContent
         super.gx_downloadFile(fileName, fileContent.getBytes()); // call super ↵
to continue with the download. Works for both ActiveX/Non ActiveX. In ActiveX mode ↵
binary content is not supported.
 }
```

.NET

In `GXBasicWebForm`:

Visual Basic

```
Public Overrides Sub gx_downloadFile(ByVal fileName As String, ByVal bytes As Byte())
        Dim encoding As New System.Text.ASCIIEncoding '' create a bytes to string ↵
ASCII converter
        Dim fileContent As String = encoding.GetString(bytes)  '' convert the bytes ↵
into String
        '' manipulate here the fileContent
        MyBase.gx_downloadFile(fileName, encoding.GetBytes(fileContent)) '' call ↵
base method to continue with the download. Works for both ActiveX/Non ActiveX. In ↵
ActiveX mode binary content is not supported.
    End Sub
```

C#

```
        public override void gx_downloadFile(string fileName, byte[] bytes)
        {
            System.Text.ASCIIEncoding encoding = new System.Text.ASCIIEncoding(); ↵
// create a bytes to string ASCII converter
            string fileContent = encoding.GetString(bytes);  // convert the bytes ↵
into String
            // manipulate here the fileContent
           base.gx_downloadFile(fileName, encoding.GetBytes(fileContent)); // call ↵
base method to continue with the download. Works for both ActiveX/Non ActiveX. In ↵
ActiveX mode binary content is not supported.
        }
```

gx_getNdtDefaultDownloadFileName

Use this user exit to manipulate/modify the name of a downloaded Natural file.

For example:

JSP

In `GXBasicContext`:

```
@Override
public String gx_getNdtDefaultDownloadFileName(String fileName, String workingFile) {

    if("D5".equals(workingFile)){
        return "NewFileName.txt";
    }
    return super.gx_getNdtDefaultDownloadFileName(fileName, workingFile);
}
```

.NET

Visual Basic

```
Public Overrides Function gx_getNdtDefaultDownloadFileName(ByVal fileName As String, ↵
ByVal workingFile As String) As String

        If ("D7".Equals(workingFile)) Then
            Return "C:\\temp\\NewFileName.csv"

        Else
            Return MyBase.gx_getNdtDefaultDownloadFileName(fileName, workingFile)
        End If

    End Function
```

C#

```
public override string gx_getNdtDefaultDownloadFileName(string fileName, string ↵
workingFile)
        {
            if ("D7".Equals(workingFile))
            {
                return "C:\\temp\\NewFileName.csv";
            }
            return base.gx_getNdtDefaultDownloadFileName(fileName, workingFile);
        }
```

GXIClientBaseObject

JSP

```
GXIClientBaseObject(getGXSession)
```

Package:`com.sabratec.applinx.baseobject` The new ApplinX base object. See **Base Object** documentation.

.NET

```
GXIClientBaseObject (gx_session)
```

Namespace: `com.sabratec.applinx.baseobject`

ApplinX Abstract Web Classes – gx Building Blocks

JSP

`GXScreenBasedJspContext`

Package: `com.sabratec.applinx.j2ee.framework.web`

ApplinX API context class, which your project context classes extend from, contains the required logic for ApplinX framework.

.NET

`GXScreenBasedWebForm`

Namespace: `com.sabratec.dotnet.framework.web`

ApplinX ASP.NET API page, which your project pages inherits from, contains the required logic for ApplinX framework.

The ApplinX abstract Web classes contain the following functions:

- gx_attach()
- gx_connect()
- gx_disconnect()
- gx_fillForm()
- gx_fillForm(GXIScreenCollection screens)
- gx_fillFormFields()
- gx_fillTable()
- gx_handleHostResponse()
- gx_handleSessionError(GXGeneralException)
- gx_syncFormWithHost()
- gx_syncHostWithForm()
- gx_prepareSendKeysRequest(string keys)
- gx_preparePathRequest(string pathName)
- gx_doSubmitKeyLogic(string keys); gx_doSubmitKeyLogic(string keys, GXCursor cur)
- gx_doSelectRowLogic(actionField); gx_doSelectRowLogic(actionField, actionValue); gx_doSelectRowLogic(actionField, actionValue, actionKey)
- gx_setField(String fieldName,GXIField field) (JSP); gx_setField(Control ctrl,GXIField field) (.NET)
- gx_isFormGenerated(String formName)
- gx_getNextFormName()
- gx_loadForm(String formName)
- gx_isFormSyncWithHost()
- gx_getForm (String screenName)
- gx_processHostKeyRequest(GXSendKeysRequest sendKeyRequest)

- gx_doCloseWindow(String hostKey)

gx_attach()

Use this function to attach to a session based on `getGXAppConfig().getSessionConfig()` (JSP)/`gx_appConfig.SessionConfig` (.NET) declarations. If the user is already attached (from the previous page), it will use the same session.

This function is called automatically when inheriting from *GXDefaultLogicContext*(JSP)/*GXDefaultLogicWebForm*(.NET) .

> **Note:** It is not necessary to detach as this is done automatically by the framework.

gx_connect()

Use this function to connect to a session based on `getGXAppConfig().getSessionConfig()` (JSP)/`gx_appConfig.SessionConfig` (.NET) declarations.

`gxfirstpage.java` (JSP)/`gxfirstpage.aspx.cs(vb)` (.NET) is the class that activates this function by default.

gx_disconnect()

Use this function to disconnect a session based on `getGXAppConfig().getSessionConfig()` (JSP)/`gx_appConfig.SessionConfig` (.NET) declarations.

`logoff.java` (JSP)/`logoff.aspx.cs(vb)` (.NET) is the class that activates this function by default.

gx_fillForm()

This function calls `gx_fillTables` and `gx_fillFormFields`. Called when the page is loaded automatically when inheriting from `GXDefaultLogicContext`(JSP)/`GXDefaultLogicWebForm` (.NET). This function is called when staying in the same screen after sending keys, when calling `gx_handleHostResponse`.

gx_fillForm(GXIScreenCollection screens)

Uses the values of the application fields in the screens of the path response to fill the page.

For example:

JSP

```
GXPathResponse res =getGXSession().executePath("collect_customer_data");
gx_fillForm(res.getScreens()); ↵
```

.NET

```
GXPathResponse res =gx_session.executePath("collect_customer_data");
gx_fillForm(res.getScreens());
```

gx_fillFormFields()

This function is called automatically by `gx_fillForm`. It fills the page with values from the current screen. The decision, which fields to update, is determined by `getGXAppConfig().getFieldTypesInUse()` (JSP) / `gx_appConfig.FieldTypesInUse()` (.NET) .

gx_fillTable()

In JSP, this function is called automatically by `gx_fillForm`. It will automatically call the `getTagsAccessor().setTagTable` method. It can be overridden in order to customize the table created to be based on a path, or to add user events.

In .NET, this function is called automatically by `gx_fillForm`. It checks if a control with the same ID as the ApplinX table is related to the current screen. If this control is an HTML table, it will automatically call the `GXTablesHandler fillHtmlTableFromGXTable` method. If the control is a DataGrid it will convert the host table into a DataTable object and bind it to the DataGrid. It can be overridden in order to customize the table created to be based on a path, or to add user events.

gx_handleHostResponse()

This function is called to synchronize between the Web application screen to the host screen. This function calls `gx_fillForm` when the screen/page was not changed, otherwise redirects to the matching page for the new host screen.

gx_handleSessionError(GXGeneralException)

This function loads the default ApplinX error page `gx_appConfig.getErrorForm`. It can be overridden to change error handling.

gx_syncFormWithHost()

This function checks to see whether the current page matches the current host screen (this function is not relevant for Instant pages). When they do not match, the framework redirects to the matching page. Use this if you want your application to work in the traditional host oriented approach.

gx_syncHostWithForm()

This function checks to see whether the host matches the current page screen (this function is not relevant for Instant pages). When they do not match, the framework calls the map declared in `getGXAppConfig().getMapName()` (JSP)/`gx_AppConfig.MapName` (.NET) , and executes it in order to navigate to the matching host screen. Use this if you want your application to work in a Web oriented approach, that is, the Web application decides what page to load (by links), and not the host.

This feature handles Web-host synchronization, and allows the ApplinX Web application to support the Back button in the browser. To implement this, it is necessary to include all the screens that are in your new Web application in the Map path.

gx_prepareSendKeysRequest(string keys)

Returns `GXSendKeysRequest` object, with the string keys typed as the argument values. Use this function to retrieve from the current page a `GXSendKeyRequest` object with all the fields from the page, and the current position of the cursor. For example, when you add a server-side button that is intended to send all fields followed by the [ENTER] key:

JSP:

```
getGXSession().sendKeys(gx_prepareSendKeysRequest("[enter]"));
```

.NET:

```
gx_session.sendKeys(gx_prepareSendKeysRequest("[enter]"));
```

gx_preparePathRequest(string pathName)

Returns `GXPathRequest` object with the `pathName`. Use this function to retrieve from the current page a `GXPathRequest` object with all the fields from the page. For example, when you add a server-side button that is intended to execute a path with all fields from the current screen and execute the path:

JSP:

```
getGXSession().executePath(gx_preparePathRequest("path_login"));
```

.NET:

```
gx_session.executePath(gx_preparePathRequest("path_login"));
```

gx_doSubmitKeyLogic(string keys); gx_doSubmitKeyLogic(string keys, GXCursor cur)

Use this function when working with server side buttons and needing to perform a simple PF key action, or a PF key action with cursor focusing. Perform the entire process of sendKeys and navigate to the next page.

`GXCursor` is in the package `com.sabratec.applinx.baseobject`.

gx_doSelectRowLogic(actionField); gx_doSelectRowLogic(actionField, actionValue); gx_doSelectRowLogic(actionField, actionValue, actionKey)

Use this function when working within a page that contains an HTML table control/tag which is bound to an ApplinX host table, and you need to perform a simple row selection action in a server-side click event.

This function performs the following process of selecting a host row in a screen that contains a host table: sends all the screen fields, selects a row in the action field (actionField), with the action value ("X", "1" etc.) and navigates to the next page.

Use the first function (`gx_doSelectRowLogic(actionField)`) when the selection column is cursor-sensitive and there is no need to place an action value in it.

Use the second function (`gx_doSelectRowLogic(actionField, actionValue)`) when an action value is required.

Use the third function (`gx_doSelectRowLogic(actionField, actionValue, actionKey)`) when the action key is not the [ENTER] key.

gx_setField(String fieldName,GXIField field) (JSP); gx_setField(Control ctrl,GXIField field) (.NET)

This function is called for each field in the screen when `gx_fillForm` is called.

JSP

This function can be overridden to affect the look-and-feel of a gx tag according to the data in the host field.

For example:

```
public void gx_setField(String tagId,GXIField field){
      super.gx_setField();  the default behavior
      if (!field. IsVisible()){
             getTagsAccesor().setTagVisible(tagId,false);
}
}
```

.NET

This function can be overridden to affect the look-and-feel of a control according to the data in the host field.

For example:

```
public override void gx_setField(Control ctrl,GXIField field){
      base.gx_setField(Control ctrl, GXIField field);  the default behavior
      if (!field. IsVisible())
             ctrl.Visible = false;
}
}
```

gx_isFormGenerated(String formName)

Returns boolean.

Determines if a page exists, so that the engine can redirect to that page.

gx_getNextFormName()

Returns string.

The name of the next form to load according to the current host screen. Can be SCREEN_NAME.jsp or instant.jsp for JSP and SCREEN_NAME.aspx or instant.aspx for .NET.

gx_loadForm(String formName)

Loads the given form. Keeps the host session open when redirecting to the target form.

JSP

Uses response sendRedirect.

.NET

Uses response redirect.

gx_isFormSyncWithHost()

Returns boolean.

Indicates whether the host screen name matches the current page name.

gx_getForm (String screenName)

Returns string.

Returns the matching page for a given host screen.

JSP

By default it is `<SCREEN_NAME>.jsp`. Can be overridable to send a few host screens to the same page.

For example:

```
public gx_getForm(String screenName){
If ( screenName.equals("SCREEN_A") || screenName.equals("SCREEN_B"){
      return "PAGE_A.jsp";
  }
  return super.gx_getForm(screenName); // for default
}
```

.NET

By default it is `SCREEN_NAME.aspx`. Can be overridable to send a few host screens to the same page.

For example:

```
Protected override gx_getForm (String screenName){
If (screenName == "SCREEN_A" || screenName == "SCREEN_B"){
      Return "PAGE_A.aspx";
}
return base. gx_getForm (screenName); // for default
}
```

gx_processHostKeyRequest(GXSendKeysRequest sendKeyRequest)

This function is activated from a browser PF key or when [ENTER] is pressed, or
`javascript:gx_SubmitKey(key).`

`gx_processHostKeyRequest(GXSendKeysRequest sendKeyRequest)`

Executes the given send keys request, and throws the relevant event:

`gx_preSendKey`

`gx_postSendKey`

This function can be overridden in order to use customized code with the relevant action key.

gx_doCloseWindow(String hostKey)

Properly closes the current host window. Use this function when a pop-up window is closed by
the user, in the `gx_closeWindow` event. Accept a parameter of a host key to send to the host in order
to close the host window.

GXIScreenBasedForm

`(gx_form - .NET )`

`(getGXForm() - JSP)`

Package: `com.sabratec.applinx.framework`

The ApplinX.NET and JSP pages are handled as the interface `GXIScreenBasedForm`. This interface
is passed to the framework engine in order to fill the fields, read the fields values and get form
details. All the interface methods are overridable and can be used to customize the page fields'
update and returned values.

- getAppFieldContent(String fieldName)
- getHostFieldContent(GXPosition pos)
- getMultipleFieldContent(String fieldName, int index)
- getFieldTypesInUse() setAppField(GXIField field)
- setHostField(GXIField field)
- setMultipleField(GXIField field)
- SeqScreenNumber
- CursorPosition
- FormName

- HostKeys

getAppFieldContent(String fieldName)

Returns to the host (from the page) the value of an AppField.

getHostFieldContent(GXPosition pos)

Returns to the host (from the page) the value of a host field by position.

getMultipleFieldContent(String fieldName, int index)

Returns to the host (from the page) the value of a multiple AppField. Only the relevant methods will be called according to the value of `gx_appConfig`.

getFieldTypesInUse() setAppField(GXIField field)

Sets the value of a page field from an AppField.

setHostField(GXIField field)

Sets the value of a page field from a host field.

setMultipleField(GXIField field)

Sets the value of a page field from AppField, overridable. The calls for the three functions depend on a call to `gx_fillForm`, and the value of `gx_appConfig.getFieldTypesInUse()`.

SeqScreenNumber

Returns the page sequential screen number.

CursorPosition

Returns the name of the field that is focused on, in the page.

FormName

Returns the current page name.

HostKeys

Returns the PF key pressed in the browser.

Instant Component

Use the following control to display the ApplinX instant HTML page. The Instant component is included in *instant.aspx/jsp* and in a generated page for a screen group. Refer to the Instant API Javadoc (by default: \SoftwareAG_documentation\webMethods\ApplinX\9-6_ApplinX\api). The instant configuration may be modified within the framework in the class `GXInstantLogicContext` (JSP)/`GXInstantLogicWebForm` (.NET). For the configuration options see the class `com.sabratec.applinx.presentation.GXRenderConfig` in the ApplinX Development API Javadoc.

Host Keys Component

The Host Keys custom tag allows developers to control the dynamic host keys collection in various common options and custom templates. This control allows creating complex UI, based on the host keys, without any code. This component is useful for creating a fast, advanced UI in a Web enabling application by combining it anywhere in the framework template. The host keys collection is rendered inside a container which may be a standard HTML table. This component can be fully customized using the `GXIHostKeysTagUserExit` interface. Refer to Customizing the Host Keys.

The custom tag contains the following attributes:

- KeyType
- cssClass(JSP)/Class(.NET)
- Vertical
- Additional Attributes

KeyType

May be Links, Buttons or Template.

Determines the way each host will be displayed. When defined to be displayed as Template, the tag's inner HTML should contain the HTML text that will be rendered for each host key. The HTML text may contain the tokens: `$(ACTION)` and `$(CAPTION)` for dynamic replacements. Action being the name of the function key itself, that will be sent to the host (for example - [ENTER], [PF5], [PA2]) and Caption, the text associated with the function key (written below or next to the function key name).

cssClass(JSP)/Class(.NET)

Determines the css class of the rendered HTML for the host keys.

Vertical

Determines how the host keys are displayed.

Possible values: `true`, `false`. When set to `false`, the host keys will be displayed horizontally.

Additional Attributes

In order to control the HTML table container of the host keys, all the standard HTML table attributes are supported: `width`, `height`, `cellspacing`, `cellpadding` and `border`.

Example of use:

Render the host keys vertically, as a custom tag in `beforemainpane`.

For example:

JSP

```
<gx_host:hostKeys vertical="true" keyType="Template" cssClass="blueButton">
<a href="#" onclick="gx_SubmitKey('$(ACTION)')" title="$(CAPTION)">
<img src="images/$(ACTION).gif">
    </a>
    </gx_host:hostKeys>
```

.NET

```
<gx:GXHostKeysControl runat="server" vertical="true" keyType="Template" ↵
Class="blueButton">
<a href="#" onclick="gx_SubmitKey('$(ACTION)')" title="$(CAPTION)">
<img src="images/$(ACTION).gif">
    </a>
    </gx:GXHostKeysControl>
```

Printer Control (.NET only)

This control generates a java applet tag for the ApplinX printlet. Refer to the **ApplinX Printer Applet**. Included within the *run_printlet.aspx* file.

Useful JavaScript Functions

- user_prePostBack
- gx_postBack(<SERVER-SIDE-FUNCTION-NAME>) (JSP only)

user_prePostBack

Occurs before a page is posted back to the server side. Can be triggered from both a server side button or keyboard PF key/ENTER. Add it to the relevant page in order to carry out validation checks.

For example:

```
<script>
function user_prePostBack(){
     if (isValid()){ // isValid() is a user function that checks validity
          return false;
}
return true;
}
<script>
```

gx_postBack(<SERVER-SIDE-FUNCTION-NAME>) (JSP only)

Used for submitting the page and activating the specified function in the associated context class. The specified function should be a public function with no parameter and no return value.

ApplinX Tables API

- Class Methods for the Table Entity - JSP
- Classes for the Table Entity - .NET
- GXITableEvents (JSP)

- GXITableEvents (.NET)

Class Methods for the Table Entity - JSP

- getTagsAccessor().setTagTable
- getTableSelectedKey(String tagId); getTableSelectedKeys(String tagId)
- addTableKeyColumn(String tagId,String keyCol)
- GXTableBuildConfig

getTagsAccessor().setTagTable

Package: `com.sabratec.dotnet.framework.web.tables`

Methods:

```
setTagTable(String tagId,GXITable gx_table);
```

```
setTagTable(String tagId,GXITable gx_table,GXITableEvents events);
```

```
setTagTable(String tagId,GXITable gx_table,GXITableEvents events, GXTableBuildConfig
config);
```

Return type: none

Description: Sets for the table TagId a `GXITable` object, for duplication by each row in the HTML table according to the data in `gx_table`.

Optional parameters:

`GXITableEvents` Events: user listener that allows customizing the HTML table at runtime, according to the host data. For example, changing the background color to red for certain rows when the content of a certain field in the current row is negative.

getTableSelectedKey(String tagId); getTableSelectedKeys(String tagId)

Returns `String/String[]`

Returns the value of the key column(s) when performing a row selection server-side event in the HTML table. Can be used to send the index to the correct multiple application fields or to activate a path that searches for the correct line by the key value.

addTableKeyColumn(String tagId,String keyCol)

Adds key column to the HTML `gx:table`. When a row is selected in the HTML `gx:table`, the primary key(s) of the row will be retrieved using the method: `getTableSelectedKey/ getTableSelectedKeys`.

GXTableBuildConfig

Description: This method is the configuration parameter for building the HTML table.

The ApplinX Frameworks also activates these APIs automatically, when inheriting from *GXBasicContext/GXDefaultLogicContext*.

Classes for the Table Entity - .NET

- GXDataConverter
- GXTablesHandler

GXDataConverter

Namespace: `com.sabratec.dotnet`

GXTableToDataTable—Static Method

Parameters: `GXITable` object

Return type: `System.Data.DataTable`

Description: Gets a GXITable object returned from ApplinX server through gx_session, and converts it to a DataTable object.

Example ©#):

Using com.sabratec.dotnet;

Using com.sabratec.applinx.baseobject.tables;

```
…
public override void gx_buildTable(){
GXITable gx_table = gx_session.getTables()[0];
DataTable dt = GXDataConverter.GXTableToDataTable(gx_table);
Datagrid1.DataSource = dt;
Datagrid1.DataBind();
}
```

GXTablesHandler

Namespace: `com.sabratec.dotnet.framework.web.tables`

fillHtmlTableFromGXTable—Static Method

`fillHtmlTableFromGXTable(HtmlTable table,GXITable gx_table);`

`fillHtmlTableFromGXTable(HtmlTable table,GXITable gx_table,GXITableEvents events);`

`fillHtmlTableFromGXTable(HtmlTable table,GXITable gx_table,GXITableEvents events, GXTableBuildConfig config);`

Return type: none

Description Gets a .NET HTML table object, and a `GXITable` object, and duplicates each row in the HTML table according to the data in `gx_table`.

Optional parameters: `GXITableEvents` Events: user listener that allows customizing the HTML table at runtime, according to the host data. For example, changing the background color to red for certain rows when the content of a certain field in the current row is negative.

This method is the configuration parameter for building the HTML table. The ApplinX framework also activates these APIs automatically, when inheriting from *GXDefaultLogicWebForm*.

> **Note:** This is an example of how to use the tables API.

Using `com.sabratec.dotnet.framework.web.tables;`

Using `com.sabratec.applinx.baseobject.tables;`

addKeyColumn—Static Method

`addKeyColumn(HtmlTable table,string keyCol);`

Description: Add key column to the HTML table. When a row is selected in the HTML table, the primary key(s) of the row will be retrieved using the method: `getTableSelectedKey/ getTableSelectedKeys`.

getTableSelectedKey/ getTableSelectedKeys—Static Method

Description: Returns the value of the key column(s) when performing a row selection server-side event in the HTML table. Can be used to send the index to the correct multiple application fields, or to activate a path that searches for the correct line by the key value.

GXITableEvents (JSP)

Package: com.sabratec.j2ee.framework.web.tables Interface for events when the HTML table rows are duplicated with run-time data.

Methods:

- gx_changeTr(int RowIndex,Element tr,GXITableRow row);
- gx_changeTd(int ColIndex, Element td,GXITableRow row);
- gx_changeControl(int ColIndex, Element td,Control ctrl,GXITableRow row);

gx_changeTr(int RowIndex,Element tr,GXITableRow row);

Occurs when each new table row (TR) with run-time data is created. Allows you to customize the TR according to the current host row.

gx_changeTd(int ColIndex, Element td,GXITableRow row);

Occurs when each new table cell TD with run-time data is created. Allows you to customize the table data TD according to the current host row.

gx_changeControl(int ColIndex, Element td,Control ctrl,GXITableRow row);

Occurs when each new control inside a TD with run-time data is created. Allows you to customize the controls according to the current host row.

GXITableEvents (.NET)

NameSpace: `com.sabratec.dotnet.framework.web.tables` Interface for events when the HTML table rows are duplicated with run-time data.

Methods:

- gx_changeTr(int RowIndex, HtmlTableRow tr, GXITableRow row);
- gx_changeTd(int ColIndex, HtmlTableCell td, GXITableRow row);

- gx_changeControl(int ColIndex, HtmlTableCell td, Control ctrl, GXITableRow row);

gx_changeTr(int RowIndex, HtmlTableRow tr, GXITableRow row);

Occurs when new table rows (TR) with run-time data, are created. Allows you to customize the TR according to the current host row.

gx_changeTd(int ColIndex, HtmlTableCell td, GXITableRow row);

Occurs when each new table cell TD with run-time data is created. Allows you to customize the table data TD according to the current host row.

gx_changeControl(int ColIndex, HtmlTableCell td, Control ctrl, GXITableRow row);

Occurs when each new control inside a TD with run-time data is created. Allows you to customize the controls according to the current host row.

Example:

```
'' in the code behind of the designed table web page
public override void gx_fillTable()
{
GXTablesHandler.fillHtmlTableFromGXTable(CustomerReportHtmlTable, ↵
gx_session.getTables()[0], this);
}

public void gx_changeControl(int colIndex,HtmlTableCell td, Control ctrl, GXITableRow ↵
 row){
// a link inside a TD will get it's value from the Subject        //  // column
if (ctrl is HtmlAnchor){
((HtmlAnchor)ctrl).InnerHtml = row.getItemContent("Subject");
    }
    }
public void gx_changeTd(int colIndex,HtmlTableCell td,GXITableRow row) {
    // a table with a red host row will get a marked row css class.
if ( ((GXFieldTableCell)row.getItem("Subject")).getFGColor() == ↵
GXBaseObjectConstants.GX_FIELD_COLOR_LIGHT_RED){
td.Attributes["class"] = "marked_row";
    }
   }

public void gx_changeTr(int rowIndex, HtmlTableRow tr, GXITableRow row){
 // a row with dashed in the Subject field will be disabled
     if (row.getItemContent("Subject").IndexOf("----") > -1){
        tr.Visible = false;
       }
   }
```

ApplinX Browser Windows API

- gx_window/getGXWindow() Methods (Client-side and Server-side)
- Pop-Up Window Configuration
- Pop-up Windows User Exits

gx_window/getGXWindow() Methods (Client-side and Server-side)

JSP:

GXIWindow (getGXWindow())

Package: com.sabratec.applinx.framework.web.windows

.NET:

GXIWindow (gx_window).

Namespace: com.sabratec.applinx.framework.web.windows

- loadPage(String pageName)
- loadPageFull(String pageName)
- close()
- open(String pageName, int Width, int height)
- addCommand(String command)
- resizeTo(int width, int height)
- refreshPage()
- cancelRefresh()
- setField(String fldName,String fldVal)
- isOpener()
- isWindow()
- moveTo(int posx, int posy)

loadPage(String pageName)

Loads the given page name in the current window, or its opener. The loading is done inside the frameset.

loadPageFull(String pageName)

Loads the given page name in the current window, or its opener. The loading is done outside the frameset.

close()

Closes the referred window(gx_window or gx_window.opener).

> **Note:** When performing a set of actions, ensure this method is the last one, because any line after it will not be executed.

open(String pageName, int Width, int height)

Opens the specified page name, with the given width and height.

addCommand(String command)

Allows you to add a JavaScript function to be executed.

resizeTo(int width, int height)

Allows you to resize the window (for example, if the host window was changed).

refreshPage()

Calls the server-side gx_refreshWindow(GXBasicContext)(JSP)/gx_refreshWindow(in GXBasicWeb-Form) (.NET) event, which can be used to update the form field.

cancelRefresh()

By default when a window is closed, the opener is refreshed (Good for instant). You can cancel this behavior in specific functions by calling: `gx_window.opener.cancelRefresh()`

setField(String fldName,String fldVal)

Allows you to return a field from the pop-up window to the main window using server or client-side code.

isOpener()

If true, means that the current window is a main window.

isWindow()

If true, means that the current window is a pop-up window.

moveTo(int posx, int posy)

Allows you to position the window (for example, if the host window was changed).

Pop-Up Window Configuration

When activating the pop-up manager, there are various configurations that the developer can configure. These can be configured in the Framework Editor, Window node.

Changing the default size of the frame columns

The pop-up manager is based on a frameset, one frame is always visible and the other isn't. In the development stage the developer may want to make the invisible frame also be visible. A typical case may be for a design time error that is not shown. To make the invisible frame visible, you are required to change the size of the frame columns in *config/gx_clientConfig.xml* `engineConfig parameters, defaultFrameColumns="100%"`, change 100% to a different suitable number.

Pop-up Windows User Exits

The following user exits will be called only when the pop-up window manager is activated:

- gx_preOpenWin
- gx_refreshWindow ( )
- gx_closeWindow ( )

gx_preOpenWin

Occurs prior to a pop-up window display, which is opened by the framework before a host window's appearance. Used for controlling if and how to open the pop-up window.

JSP

Use `event.setWidth (jsp), event.setHeight` to control the size of the opened window.

Use `event.setPosX(jsp), event.setPosY(jsp)` to control the position of the opened window.

Use `event.setLoadAsWindow(false)` to cancel the opening of the pop-up window (will be opened as a main page).

.NET

Use `e.Width, e.Height` to control the size of the opened window.

Use `e.PosX, e.PosY` to control the position of the opened window.

Use `e.LoadAsWindow = false` to cancel the opening of the pop-up window (will be opened as a main page).

gx_refreshWindow ( )

Occurs in the main window page whenever the user closes a pop-up window using the X button. Allows the developer to update the page when this event occurs. By default, calls `gx_fillForm`.

gx_closeWindow ( )

Occurs in the pop-up window page whenever the user closes a pop-up window using the X button. Allows the developer to send an exit key (PF3 for example) to the host so that the Web application will be synchronized with the host application. Typical usage: `gx_doCloseWindow("[PF3]")`.

JSP API

Refer to the JSP tags API. To view the Javadoc Development API,locate within your ApplinX installation the **api** folder (by default: \SoftwareAG_documentation\webMethods\ApplinX\9-6_ApplinX\api).

26 Client Side (JavaScript)

ApplinX Server Actions

Refer to the following tasks for further details on implementing these functions:

- Creating a Button / Hyperlink for Submitting a Host Key
- Enabling Sending Dup and FieldMark Characters to the Host

gx_SubmitKey(key)

Submits a PF key to the host.

parameters

key
   The key to submit.

   Example:

```
"[ENTER]", "[PF3]"  ↵
```

gx_SubmitKeyInPos(pos, keyName)

Sets the position of the cursor and then submits a PF key to the host.

Parameters

pos
   The host screen position from which to send the host key. In a standard 24X80 host screen, the position can be any number between 1 and 1920.

keyName
   The host key to send to the host.

   For example

```
"[ENTER]"
```

SubmitCustomKey()

Prompts the user to specify a host key (for example "[PF3]") and submits it to the host.

gx_SetCursorPos(pos)

Sets the position from which the next PF key will be sent. In a standard 24X80 host screen, the position can be any number between 1 and 1920.

gx_ExecPath(pathName)

Executes an ApplinX Path Procedure for navigation purposes. When a folder is not specified, ApplinX server looks for the path in the directory of the current host screen. If the path cannot be found in the directory of the current host screen, ApplinX server looks for the path in the root directory. When a folder is specified, it needs to be relative to the root directory.

Parameters

pathName
The path to execute (case-sensitive).

For example:

```
ExecPath("gotoMainMenu")
ExecPath("Common/gotoMainMenu") ↵
```

gx_systemRequest()

Prompts the client to type in the system request and then submits it to the host. Equivalent to gx_SubmitKey("[sysreq]").

gx_fieldmark()

Places the Field Mark symbol in the currently selected input field, and moves the cursor to the next unprotected position. Handling of the symbol is dependent on the application program. Equivalent to gx_SubmitKey("[fieldmark]").

gx_dup()

Simulates pressing the MainFrame duplicate (Dup) key. A duplicate symbol ("*") will appear in the currently selected input. Upon submitting the form, APX server will send the dup command. This only works on host fields that allow duplicating (in the host application). The host should support this functionality as well.

Navigating between Input Fields

Refer to the following task for further details on implementing these functions:

■ Navigating between Input Fields

gx_home()

Moves to the first input element on the page. Equivalent to gx_SubmitKey("[home]").

gx_end()

Moves to last input element on the page. Equivalent to gx_SubmitKey("[end]").

gx_newLine()

Moves to the first input field you find in the rows below. Equivalent to gx_SubmitKey("[newline]").

gx_jumpToNextInput(currTextBox)

Sets the cursor on the next input field on the screen, relative to currTextBox.

For example:

```
<input type="button"
     onclick="gx_jumpToNextInput(this)"… ↵
```

gx_jumpToPrevInput(currTextBox)

Sets the cursor on the previous input field on the screen, relative to currTextBox

For example:

```
<input type="button"
onclick="gx_jumpToPrevInput(this)"… ↵
```

Tables

Refer to the following tasks for further details on implementing these functions:

- Retrieving Values from a Selected Row within a Table

gx_selectKey(elem)

Each table in a generated page keeps track of the selected row in the HTML table by a hidden field, which is added automatically.

This JavaScript function, when called inside a table row, updates this hidden field with the row number (by default) or row key to keep track of the selected row on the server-side.

Can be inserted by an on click event on a table row (tr) or a link/button placed inside a table cell.

Parameters

elem
Any element inside a table row (including the TR element itself).

gx_getSelectedKey(tableName)

Returns the selected key after `gx_selectKey` was called.

Parameters

tableName
The name/ID of the HTML table whose selected value we want to retrieve.

gx_isTableKeySelected(tableName)

Query if a row was selected. Used for validation to force a row selection.

Possible values: true/false

parameters

tableName
    The name of the HTML table.

gx_markRow(obj,selectedRowCss)

Can be called in a table row (tr) event. This function marks the clicked row with the selected row css class name.

Can be used by ApplinX css: gx_markRow(this,"gx_tbl_selected")

Parameters

obj
    The table row object selected.

RowCss
    The css class to use to change the selected row.

Design

Refer to the following tasks for further details on implementing these functions:

■ Enabling the User to Control the Font Size

gx_changeCss()

Toggles between the display styles specified in the web application configuration (In Emulation node>Color Set).

gx_changeCssExact(cssName)

Sets the display style to the specified style sheet.

Parameters

cssName
　　Style sheet URL.

gx_increaseFontSize()

Increases the font size used in the Web application. This function is called by the plus link in the page footer. The maximum font size is 20px. Refer to Enabling the User to Control the Font Size.

gx_decreaseFontSize()

Decreases the font size used in the Web application. This function is called by the minus link in the page footer. The minimum font size is 7px. Refer to Enabling the User Control the Font Size.

gx_changeFontSize(size)

Changes the font size used in the Web application to the specified size.

Parameters

size
　　An integer. The new font size in pixels.

Keyboard Mapping

Refer to the following tasks for further details on implementing these functions:

■ Mapping Keyboard Keys to User Actions in Individual Pages

gx_AddKeyboardMapping(additionalKey,keyCode,functionElement,overrideExisting,cancelMapFunction)

This function allows developers to attach JavaScript functions to specific keyboard keystrokes.

Parameters

additionalKey
　　Possible values : 0-none, 1- CTRL, 2 – ALT, 3- SHIFT

keyCode
　　An integer. The ASCII code of the pressed key. For example: Enter = 13

functionElement

    The JavaScript element to execute.

overrideExisting

    A boolean parameter indicating whether to override the existing XML keyboard mapping definition (gx_keyboardmapping.xml) .

cancelMapFunction

    Optional. A function element that returns either true or false, determining whether to execute the keyboard mapping function or not. True, cancels the mapped function and False executes the function.

Examples

When CTRL+ESC are pressed, executes a confirm `function(myConfirmFunc)`. When confirmed, performs the `myLogoff` function.

```
gx_AddKeyboardMapping(1,27,myLogoff,true,myConfirmFunc)
```

When CTRL+ESC are pressed, executes the `myLogoff` function.

```
gx_AddKeyboardMapping(1,27,doLogoff,true)
```

Overrides the default PF3 action with a do-nothing action (`void(0)`).

```
gx_AddKeyboardMapping(0,27,void(0),true)
```

ApplinX Web Application Event

What is GXEvent?

GXEvent is a Wrapper for a browser event. This wrapper provides a cross browser event object.

Refer to the following task for further details on implementing these functions:

- Implementing & Controlling JavaScript Events using the gx_event Object

Properties

GXEvent.keyCode

Sets or retrieves the Unicode key code associated with the key that caused the event.

GXEvent.additionalKey

An integer representing the key that was pressed in addition to the key that triggered the event. Possible values: 0 – none, 1-CTRL, 2-ALT, 3-SHIFT

GXEvent.element

Retrieves the element that fired the event.

Methods

GXEvent.cancel()

Cancels the event and stops it from bubbling further up the hierarchy of event handlers.

ApplinX Web Application Event Example

The following example will cancel the `OnKeyDown` whenever the [ENTER] key is pressed in a specific text area ("myTextArea"), prevent the page from being submitted and manually add a newline to the text area value:

Add the following to `globalOnKeyDown` function in the userExits.js file:

```
function globalOnKeyDown(gx_event){
.........
 var win = gx_event.window;

 if (gx_event.keyCode==13 && gx_event.element.id=="myTextArea"){
  gx_event.cancel();
  GXBrowserUtil.getElement("myTextArea").value += "\r\n"
 }
.........
}
```

Assume your JSP/ASPX page has the following input: `<textarea row="5" id="myTextArea" ></textarea>`

Browser Related Functions

Refer to the following task for further details on implementing these functions:

■ Retrieving Browser Information

GXBrowserUtil.isIE()

Boolean. Retrieves whether or not the browser is Microsoft Internet Explorer.

GXBrowserUtil.isIE7()

Boolean. Retrieves whether or not the browser is Microsoft Internet Explorer 7 or higher.

GXBrowserUtil.isMozilla()

Boolean. Retrieves whether or not the browser is Mozilla FireFox.

JavaScript Logging

GXLog is used to handle client side JavaScript logging. While mostly used by the ApplinX web application JavaScript engine, users can also add their own log messages. Refer to JavaScript Logger Engine for further details.

GXLog.debug(moduleName, message)

Adds a log message only when in debug mode.

Parameters

moduleName
   This parameter can be an empty string. However, in order to make log messages more distinctive in the log, it is recommended to add a unique value to the `engineConfig debugModules` property in *config/gx_clientConfig.xml* and then use that value as the `moduleName` for debugging the JavaScript.

message
   The log message.

GXLog.warning(moduleName, message)

Adds a log message only when in Warning mode.

GXLog.error(moduleName, message)

Adds a log message only when in Error mode.

Page Validation

GXValidator represents an array of validators, used to validate the form's data before submitting it to the host.

Refer to the following task for further details on implementing these functions:

■ Validating your Data

GXValidator.registerValidator(Validator)

Adds a validator to the array.

GXValidator.clearValidators()

Clears the validator array.

Page Validation Example

The following example is automatically added to every JSP/ASPX page generated by ApplinX:

```
var pageValidator = new function(){
 this.validateField = function(inputField){
  if (inputField.name == "FIELD_A"){
   if (inputField.value == ""){
    return "Field cannot be empty";
   }
  }
  // ...
 }
}
function pageOnLoad(){  // register validator function
 GXValidator.registerValidator(pageValidator);
}
```

This validator checks that "Field_A" has a value. Otherwise, it returns an error message.

ApplinX Web Application Windows

What is gx_window JavaScript class?

GXWindow interacts with ApplinX framework's server side gx_window(.NET)/getGXWindow() (JSP). In addition, gx_window can perform actions on the browser side using the following methods:

Refer to the following task for further details on implementing these functions:

■ Handling Web Application Windows using the gx_windows Object

gx_window.open(page,width,height,left,top)

Locks the current main window, opens a new pop-up window and loads the document specified by a given URL (page).

Parameters

page
　　A page name (including relative folders) to load into the pop-up window.

width
　　The window's width.

height
　　The window's height.

left
　　An integer. The position of the window's left edge, in relation to the desktop.

top
　　An integer. The position of the window's top edge, in relation to the desktop.

gx_window.resizeTo(width,height)

Sets the size of the window to the specified width and height values.

gx_window.moveTo(posX,posY)

Moves the screen position of the upper-left corner of the window to the specified `posX` and `posY` position.

gx_window.setField(fldName,fldVal)

Sets the value or InnerHTML property of field `fldName`, to the specified `fldVal`.

gx_window.close()

Closes the window. If the main window is closed the host session will be disconnected.

gx_window.getOpener()

In pop-up windows, retrieves the parent window element.

gx_window.loadPageFull(PageName)

Loads a URL. The PageName represents the URL of the document that is to be loaded.

gx_window.loadPage(PageName)

Loads a URL. The PageName represents the URL of the document that is to be loaded.

ApplinX Web Application Windows Example

Main Window HTML and Script

```
<script>
    function getData(){
      gx_window.open("myData.jsp",200,300,0,0) ;

  //  .NET
  // gx_window.open("myData.ASPX",200,300,0,0);
    }
    </script>
<input id="myInput"  value=" "/>
<a href="#" onclick=" getData ();">open Win</a>
```

Pop-Up Window

```
function setMainWindowInput(){
          // Set the value of an Input on the main window
              gx_window.getOpener().
                  gx_getElement("myInput").value="someValue";
              // Unlock the main window before closing the pop up
        gx_window.getOpener().gx_unlockScreen();
              // Close the pop up window
        gx_window.close();
    }

<a href="#" onclick=" setMainWindowInput ();">close Win</a>
```

HTML Controls

gx_updatePagePart(panelId)

This functionality can only be used once this feature has been selected in the Framework Config-uration Editor. This feature is used for updating a specific part of the page (can only be used once within a single page). This is useful when sorting/scrolling within host tables or working with tabs. Call this function just before a submitting a key to the host.

Parameters

panelId
The server side control ID, whose content we wish to update.

gx_isValidInputElement (inputElement)

Boolean. Returns "true" if `inputElement` is either a combo box, text area or an input tag of the fol-lowing type: radio, check box, password or text. Otherwise returns "false".

Parameters

inputElement
The input field to validate.

gx_getElement(elemId)

Similar to `document.getElementById("id")`, this function retrieves the required HTML object from the active frame according to its identifier (`elemtId`).

gx_eraseEOF()

Removes text from an input field, leaving only the text before the cursor.

gx_showCalendar(Title,dateFieldName1,Format1,dateFieldName2,Format2, dateFieldName3,Format3)

Displays the calendar window, attaches it to a specific input or inputs (up to three inputs: day, month and year) and sets the format the selected date will be displayed with.

Parameters

Title
   Calendar window title.

dateFieldName1/dateFieldName2/dateFieldName3
   An Input field used to display the date (or part of it).

Format1/Format2/Format3
   The format in which the date will be displayed.

   The fields dateFieldName2/dateFieldName3 and Format2/Format3 are optional and required only when there is more than one date field and date format.

Examples

```
<img onclick="gx_showCalendar('Select a date','BirthDate','MM-dd-yy')"… ↵
```

```
<img onclick="gx_showCalendar('Select a date','day','dd' ,month,'MM', 'year','yyyy')…
```

Possible date format examples:

MMM, d, yyyy - March 22, 2009

dd/MM/yy - 22/03/09

MM-yyyy - 03-2009

Web Application Configurations

gx_getCookie(name)

Retrieves the value of the cookie <name>.

Parameters

name
> The name of the cookie.

> **GXLAFHandler.COLOR_CSS_COOKIE**
>> Saves the user preferred Style sheet.

> **GXLAFHandler.FONT_SIZE_COOKIE**
>> Saves the user preferred font-size for Instant pages

> **GXLAFHandler.RESOLUTION_COOKIE**
>> Saves the user's screen width (this is useful when developers set the Instant configuration's font-size node to "Dynamic by screen resolution").

gx_setCookie(name, value)

Sets the value in a specific (<name>) cookie.

Parameters

name
> The name of the cookie.

> **GXLAFHandler.COLOR_CSS_COOKIE**
>> Saves the user preferred Style sheet.

> **GXLAFHandler.FONT_SIZE_COOKIE**
>> Saves the user preferred font-size for Instant pages

> **GXLAFHandler.RESOLUTION_COOKIE**
>> Saves the user's screen width (this is useful when developers set the Instant configuration's font-size node to "Dynamic by screen resolution").

value
> The value to be set to the cookie.

Functionality

Refer to the following tasks for further details on implementing these functions:

- Printing a Capture of the Host Screen
- Opening the File Transfer Dialog Box
- Deploying the Printlet

gx_printScreen()

Opens a customized print screen Web page that contains a text representation of the current host screen. This is used in order to print a snapshot of the current host screen.

gx_openFtpDialog

Opens the file transfer dialog box.

gx_openNewBrowser (url)

Opens a new browser window with its own process. Useful when you need to open multiple sessions from the same Web application. The ApplinX emulation component ActiveX Object in the current page needs to be uncommented (see *index.jsp/aspx* page in the HTML emulation demo).

Parameters

url
    The URL destination of the new window.

gx_hidePrinter/gx_showPrinter()

Hides/shows the Printlet window/frame. Will only work if the ApplinX application is printer enabled. Refer to Deploying the Printletfor more details about ApplinX printers.

gx_postBack(func)

Use this function to execute server side functionality.

Parameters

func
    A string. The server side function name to be executed.

Example

Create a button in your JSP page that when clicked runs a server side function called `myServersideFunc`:

```
<gx:input type="button" value="GO!" onserverclick=" myServersideFunc" />
```

The server side code must have a method like this:

```
public void myServersideFunc(){
     .........
}
```

Screen Locker

The ApplinX framework contains a built-in feature of a screen locker. The purpose of a screen locker is to indicate to the user by means of a message, that the application is processing his request, and blocks him from interfering with the current process by repressing a button/link or keyboard PF/ENTER.

Refer to the following tasks for further details on implementing these functions:

- Activating the Screen Locker
- Handling the Screen Locker on the Page Level

gx_lockScreen(); gx_unlockScreen()

Locks/unlocks the screen locker. Can be used from client-side code to activate/cancel the screen locker. When the screen locker is locked, no additional host keys/links/buttons can be pressed until the screen locker is unlocked.

gx_disableScreenLocker(); gx_enableScreenLocker()

Disables/Enables the screen locker ("Please wait" message) before/after redirecting to a new page.

Usage: When the target page downloads a file, screen locking is not needed. Without using the following functionality the current page will get "stuck" with the "Please wait" message, which causes a lock on the page.

```
// Disable the screen locker
    gx_disableScreenLocker();
    // redirect to the excel download page, which opens a save/open/cancel window ↵
dialog
    location.href = "excel.jsp"; // or "excel.aspx"
    // Enable the screen locker after 2 seconds for the next page actions
    window.setTimeout(gx_enableScreenLocker,2000);
```

User Exits

The user exits file is located in the js directory of your web application. It allows you to add functionality to your pages without disrupting the framework's events.

Registering Events

gx_engine.registerEvent

Registering an event to a page means that the function is executed when the event bubbles up to the document level.

```
gx_engine.registerEvent(GXEventType.FOCUS,globalOnBlur);
```

gx_engine.attachInputTagsEvent

Registering an event to an input means that the function is executed when the input fires the event:

```
gx_engine.attachInputTagsEvent (GXEventType.FOCUS,  globalInputOnFocus);
```

gx_engine.setLabelFocusFunction

This function sets the LabelFocus function:

- labelFocus: This function is executed when the label is clicked.
- labelBlur: This function is executed once the label has been clicked once and then another object became in focus.

```
gx_engine.setLabelFocusFunction(labelFocus,labelBlur);
```

Global Functions

Code placed in these global functions affects the whole application and therefore must be used carefully.

For example (taken from userExits.js):

```
function globalOnLoad(gx_event){
     // use win.<SOMETHING> to access the page tags
     // for example: win.document.GX_form
     var win = gx_event.window;

     // activate page scope function if exits
     activateIfExists(gx_event,gx_event.window.pageOnLoad);

     .........
}
```

Notice that every global function contains the line:
`activateIfExists(gx_event,gx_event.window.functionName)` which executes the user functions: <functionName> (pageOnLoad in this case), if indeed the function exists on the page. This enables running the mentioned function (in the example above: pageOnLoad) in a specific page.

For Example: Adding the following function to a generated JSP/ASPX in the ApplinX web application will cause this function to automatically execute whenever the page is loaded and pop-up the alert message:

```
function pageOnLoad(){
    alert("Hello World!!");
}
```

Available User Exits

Global function	Page level function	Description
globalOnLoad	pageOnLoad	Occurs immediately after a page is loaded.
globalOnKeyDown	pageOnKeyDown	Occurs when a keyboard key is pressed.
globalOnKeyPress	pageOnKeyPress	Occurs when the user presses an alphanumeric key.
globalOnKeyUp	pageOnKeyUp	Occurs when a keyboard key is released.
globalOnFocus	pageOnFocus	Occurs when an object gets focus.
globalOnBlur	pageOnBlur	Occurs when an object loses focus.

27　HTML Emulation

Every framework folder (JSP/.NET) contains the emulation template folder, which is based on the new application. The emulation template turns on all the emulation behavior and contains css files for different resolutions and color sets.

Install the emulation template in the same way as the ApplinX new Application in the relevant environment: JSP/.NET. Refer to Creating an ApplinX Web Application.

Default Keyboard Mapping

The following keyboard list is the default mapping from the keyboard to host keys when installing a new Web application:

F1-F12	PF1-PF12
SHIFT+F1-SHIFT+F12	PF13-PF24
Page Up, Page Down	Pageup, pagedn

All hosts

CTRL+F1-CTRL+F12	PA1-PA12
Page Up	PF7
Page Down	PF8
ESC	Attention

Mainframe only

+	Field Plus
ESC	Attention
SHIFT+ESC	SYSREQ (System request)

AS/400 only

F1-F12	P1-P12
SHIFT+F1- SHIFT +F8	P13-P20

BS2000 only

28 Printing

Printer Session API

ApplinX supports printer sessions only on AS/400 and mainframe hosts. It connects to the host, retrieves the print buffers and analyzes them. The host handles the printer's queue and the connection of printer sessions to display sessions. ApplinX connects to the host as a printer session to receive the print buffers and allows you to work with them.

The printer solution of ApplinX has two implementations. The first is by way of the Application Programming Interface (API), which reflects the data returned into the programming environment. From there, the data is sent to databases, email, Web or desktop applications.

The second option is the default behavior of an emulator, where ApplinX sends all print jobs to the client's machine by means of a printer applet that runs on the client's browser.

To view the Javadoc Development API, locate within your ApplinX installation the **api** folder (by default: \SoftwareAG_documentation\webMethods\ApplinX\9-6_ApplinX\api).

ApplinX Printer Applet

ApplinX supports printer sessions only on AS400 and mainframe hosts. It connects to the host, retrieves the print buffers and analyzes them. The host handles the printer's queue and the connection of printer sessions to display sessions. ApplinX connects to the host as a printer session to receive the print buffers and allows you to work with them.

The printer solution of ApplinX has two implementations. The first is by way of the Application Programming Interface (API). The second option is the default behavior of an emulator, where ApplinX sends all print jobs to the client's machine by means of an applet that runs on the client's browser. The applet runs on the supported browsers using the **Java Plug-in Technology**. It is a signed applet, because it is Java code and it invokes the client machine's print dialog, which requires permissions that exceed regular applet permissions. Once the applet is signed it can only address a server where the applet originated, so ensure that the Web server and ApplinX server are on the same machine or failing this, use the ApplinX Redirector .

Printing

Printer Applet Parameters

The printer applet receives parameters. The following lists describe the parameters and a brief description of each. They are all optional except for serverURL, application and device name/associate device name (mainframe only).

The printer applet is activated using the page *run_printlet.aspx* (default). The page's name can be changed, or the printer control (which generates an applet tag) from it can be moved to any other page.

- Connection Parameters
- Specific Printer Device Parameter
- Print Mode
- Print to File
- Print Data Retrieval Parameters
- Applet Parameters
- Buffer Analysis Parameters
- Data Stream
- Print Format Parameters
- Fit-to-Page Format Parameters

Connection Parameters

The connection parameters customize the connection to the ApplinX server.

Connection Parameter	Description
userid	The user's ID (obligatory).
password	The user's password. The password that connects the user to the ApplinX server.
description	A textual string describing the ApplinX SessionID attached to the instance of the ABO. Helpful to distinguish users in the ApplinX Administrator.
serverURL	The method is set by setServerURL("applinx://<serveraddress>:<serverport>") Secured SSL socket: setServerURL("applinxs://<serveraddress>:<serverport>") Note: Memory sharing method is not suitable.
application	The ApplinX application name. By default, it is the default application of the ApplinX server.
device_name	The name of the printer device as defined in the host.
associate_device_name	The name of the display device (on the host) that the printer should connect to. Note: Applicable for Mainframes only. The host handles the linkage of printer session to the display device.
message_queue	The name of the queue where messages about printing are sent. Note: Applicable for AS400 only.

Connection Parameter	Description
message_lib	The name of the library where messages about printing are sent. Note: Applicable for AS400 only.
host_address	The host address where the printer device is defined.
host_port	The host port that the printer device is defined to work with.
auto_reconnect_number_of_tries	The number of times to try to reconnect to ApplinX server.
auto_reconnect_time_interval	The interval between each attempt to reconnect to the ApplinX server.
disconnect_previous_session	Disconnects the previous session and creates a new session in cases where the network was disconnected and the user was still connected to ApplinX server.

Specific Printer Device Parameter

The Specific Printer Device parameter printer_device, allows you to set the applet to print silently to a specific printer without invoking a print dialog first. Specify one of the following values:

The value "<default>" indicates that the applet should print to the default printer silently.

The value "<First Selected Printer>" indicates that the Print dialog is displayed the first time you print in the current session. The applet will print silently to this printer in additional print jobs in the same session.

The value of the parameter is the name of a specific printer. This printer must appear on the list of available printers on the computer where the applet will run.

As the page that invokes the applet is a Web page, the same page is executed for all users. If you require different settings for selective users, change the parameter value accordingly.

Print Mode

The print mode determines whether the printing layout is graphical or defined by the printer. When defined by the printer, all Print Format and Fit-to-Page Format parameters are ignored. The layout must be defined by the printer when the print data includes transparent commands. The print mode feature is only supported when using Microsoft JVM.

bypass_gdi	Possible values are False (default), indicating that the printing layout is graphical or True, indicating that the printer will define the layout.

Print to File

Print to File Parameter	Description
output_file_name	The file name to which you want to print the job. It is possible to add a time stamp to the file name (%t), creating a new file for each print job.
append_to_file	When True, print jobs in the same session are added to the same file. By default false.
printjob_separator_text	When using the options print_as_text and append_to_file, a few print jobs may be printed to the same text file. When this happens, the text set to the parameter "printjob_separator_text" will be printed to the file between consequent print jobs.
printjob_separator_add_new_line	When using the option to append_to_file, a few print jobs may be printed to the same text file. When this happens, they can be separated by adding a new line between consequent print jobs.
print_as_text	Determines whether the job will be printed to a file (true) or to the printer (false-default).

Print Data Retrieval Parameters

Print Data Retrieval Parameter	Description
wait	The time to wait for each job to end (default 1 minute).
sleep	The time in between requests to determine the end of a print job.

Applet Parameters

Applet Parameters	Description
loglevel	Advises the applet how much and what to write to the log. Values can be normal (default) or debug.
enable_presentation	Setting the value to true displays the graphic user interface of the applet. If it is false, the log of the applet can still be viewed via the browser Java console (default: true).

Buffer Analysis Parameters

Buffer Analysis Parameters	Description
lines_per_page	Sets lines per page. Usually, set according to the host's definition (default value equals 66 or if value is set to "0"). However, you can override this according to your requirements. Page is cut when it reaches the number of lines per page.
characters_per_line	Sets characters per line. Usually, set according to the host's definition (default value equals 80 or if value is set to "0". However, you can override this according to your requirements. Line is cut when it reaches the number of characters per line.

Buffer Analysis Parameters	Description
exclude_last_page_if_empty	It is possible to define that when the last page of a print job is blank, this page will not be printed.
exclude_last_line_if_empty	It is possible to define that when the last line of a print job is blank, this line will not be printed.
do_not_add_form_feed_at_end_of_job	By default, the value of this parameter is false, printing the blank page. By default at the end of a printing job, the Printlet sends the from feed command to the printer. To disable this option, set this parameter to TRUE.
do_not_print_siso_as_space	Determines whether to print Shift-In and Shift-Out as spaces. Possible values: true, false. By default: TRUE.
treat_carriage_return_as_new_line	Treats carriage return as a new line. By default, set to FALSE.
Do_not_return_print_buffers	The contents of the print job will not be saved to memory, therefore it will not be possible to retrieve the contents using getOriginalBuffer. By default the contents are saved to the buffer (FALSE). Change the value to TRUE so as not to save the contents.

Data Stream

Throughout the printed data, transparent commands may appear. The applet is capable of identifying these transparent commands (this feature is applicable to LU3, 3270 protocol). These transparent commands, such as PCL commands, are analyzed by the front end printer and are identified by the textual characters which appear at the beginning and end of these commands (the textual characters can be one or two characters). The following two parameters are used to identify the textual characters. When these parameters are not defined, and the data includes transparent commands, the commands will be printed as part of the regular data. In order for the commands to be sent to the printer, bypass_gdi parameter must be set to TRUE.

Parameter	Description
transparent_start_trigger	Defines one or two characters used to identify the beginning of the transparent commands.
transparent_end_trigger	Defines one or two characters used to identify the end of the transparent commands.

Print Format Parameters

Print Parameters	Description
font_name	Default font name is Courier New.
font_size	Default font size is 10.
font_bold	Default is not bold (false).
font_italic	Default is not italics (false).
CPI(characters per inch), LPI(lines per inch)	Any negative number results in the best-fit print job. Density is determined according to the font size. Zero value sets CPI/LPI values as defined in the host (default). If dynamic settings are on, the value of CPI/LPI parameters is ignored. Positive number results in a manual fixed value, as follows: LPI: 4, 5.3, 6, 6.3, 8, 8.5, 9.6, 12, 24, 48 CPI: 5, 10, 12, 13.33, 15, 17.14, 20, 26.66
left_margin	Set margins for a page (default = 0).
top_margin	Set margins for a page (default = 0).

Fit-to-Page Format Parameters

Sometimes print jobs can vary between portrait and landscape format, or the host settings may not fit the current printers used by the end users. So you may use dynamic settings to squeeze print jobs into the correct page dimensions. Dynamic settings include the changing of the font and CPI/LPI settings according to the number of rows and columns in a print job.

By default, the font size is set by the parameter font_size. If the parameter change_font_size_by_chars_per_line is set to true, the printer applet dynamically reduces the font size (if necessary) for every print job, according to the number of characters in the longest line of the first page. The minimum size of font is set by the parameter minimal_font_size. If the minimal font size still doesn't fit, and the parameter change_orientation_by_chars_per_line is set to true, the printer applet sets the orientation to Landscape, and sets the font back to font_size. If it still does not fit, it reduces the font size again until it fits the page.

The parameter check_each_page_separately indicates whether the font should be checked separately for each page in the print job.

Parameters	Description
orientation_type	Forces specific orientation: portrait or landscape (default: use_host (applies the host orientation definitions)). This option does not function in MSJVM.
change_font_size_by_chars_per_line	Searches for the longest line on the page, checks if the characters in the line fit into the page dimensions. If negative, will change font accordingly. If minimal_font_size is empty, the applet will not reduce the font size (default: false). Ensure that you defined the characters_per_line parameter.

Parameters	Description
change_orientation_by_chars_per_line	Automatically changes the orientation to landscape when text does not fit in portrait page (default: false). Ensure that you defined the characters_per_line parameter.
minimal_font_size	The minimum size of font is set by this parameter. Must be set in order for change_font_size_by_chars_per_line to work.
check_each_page_separately	Indicates whether the font should be checked separately for each page in the print job (default: false).
right_margin	When you set the right margin, you limit the printable area of the page. If you use dynamic font settings then the applet will squeeze in within the printable area. Note: When dynamic font settings are disabled, the font size and CPI/LPI are fixed values. The right margin is ignored as these fixed values take precedence.
bottom_margin	When you set the bottom margin, you limit the printable area of the page. If you use dynamic font settings then the applet will squeeze in within the printable area. Note: When dynamic font settings are disabled, the font size and CPI/LPI are fixed values. The right margin is ignored as these fixed values take precedence.
best_fit_row_proportion	Percentage of font size, to use as best fit row proportion. Used in cases where a print job has a relatively small number of lines. The printlet will increase the proportion of the row gradually (so it still fits the page). Example value: 130%.