Server Side API (Java/.NET)

General

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_isSupportedFeature
- isUsingactiveX()

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);
        }
```

gx_isSupportedFeature

Method `gx_isSupportedFeature` indicates whether a particular feature is supported by the browser. Currently the only supported feature IDs are:

- `html.inputtypes.date` for the HTML5 calendar feature
- `html.inputtypes.month` for the HTML5 calendar feature with month format
- `browser.window.support` for modal window support on mobile devices

Additionally, a tag is available under Java and .NET to determine whether a feature is supported. Sample code for calling the method and examples of using the tag are given below.

JSP

Method example:

```
try {
        boolean isSup = gx_isBrowserSupportingFeature(GXIBrowserSupportedFeatures.HTML_INPUT_TYPE_DATE);
    } catch (GXUnsupportedFeatureException e) {
        GXLog.debug(this, "Feature detection not already initialized...");
    }
```

Tag example:

```
<gx:isFeatureSupported id="html.inputtypes.date" attributeName="dateSupported"/>
<%
    Boolean feature = (Boolean)request.getAttribute("dateSupported");
    if (feature != null ){
        out.print("Browser is supporting feature 'html.inputtypes.date' : "+ feature);
    } else {
        out.print("Feature detection not already initialized...");
    }
%>
```

or more simply:

```
<%=dateSupported %>
```

.NET

Visual Basic

Method example:

```
Try
    Dim sup As Boolean

    sup = gx_isBrowserSupportingFeature("html.inputtypes.date")

Catch ex As Exception
    GXLog.debug(Me.GetType(), "Feature detection not already initialized...")
End Try
```

Tag example:

```
<%@ Register Assembly="GXDotnet" Namespace="com.sabratec.dotnet.framework.web.controls" TagPrefix="gx" %>

<gx:GXIsFeatureSupported runat="server" featureID="html.inputtypes.date" attributeName="isSupportDateHTML5" />

<%
     Dim res
          res = HttpContext.Current.Items("isSupportDateHTML5");
          if res IsNothing Then
          Dim feature As Boolean
               feature = CType(res,Boolean)
               Response.Write("Browser is supporting feature 'html.inputtypes.date' : "& feature)
          Else
               Response.Write("Feature detection not correctly initialized...")
          End If

%>
```

or more simply:

```
<%=dateSupported %>
```

C#

Method example:

```
try
   {
        bool b = gx_isSupportedFeature("html.inputtypes.date");
   }
   catch (Exception ex)
   {
        GXLog.debug(this.GetType(), "Feature detection not already initialized...");
   }
```

Tag example:

```
<%@ Register Assembly="GXDotnet" Namespace="com.sabratec.dotnet.framework.web.controls" TagPrefix="gx" %>

<gx:GXIsFeatureSupported runat="server" featureID="html.inputtypes.date" attributeName="isSupportDateHTML5" />

<%
         Boolean feature = (Boolean) HttpContext.Current.Items["isSupportDateHTML5"];

                if (feature != null ){
                        Response.Write("Browser is supporting feature 'html.inputtypes.date' : "+ feature);
                } else {
                        Response.Write("Feature detection not correctly initialized...");
                }
%>
```

Or more simply:

```
<%= HttpContext.Current.Items["isSupportDateHTML5"]%>
```

isUsingactiveX()

Use this user exit to manipulate/modify the name of a downloaded Natural file. To implement the method `gx_getNdtDefaultDownloadFileName` for automatic download, use the property `isUsingactiveX()`. This property indicates if the process is using ActiveX or not, and changes the logic accordingly.

For example:

```
public String gx_getNdtDefaultDownloadFileName(String filename, String workingFile) {

     if (workingFile.equals("D7")){
     return "C:\\temp\\MyFile.csv";

}
     if (!isUsingActiveXControl()){
     return "Natural_"+workingFile+".csv";
     }
     return super.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 in the ApplinX API Specification. 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

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

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

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 its 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 GXBasicWebForm) (.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 in the ApplinX API Specification.