

Migrating Applications to General Interface 3.2

This document contains a list of tasks to complete for migrating General Interface applications to 3.2. However, this list might be incomplete and will be updated as other issues are known.

This document does not describe features and APIs from 3.1 that have been deprecated in 3.2. See the Release Notes for more information.

Version 1.0. 11-21-2006

Scope: Version 3.2.0



<http://www.tibco.com>

Global Headquarters

3303 Hillview Avenue
Palo Alto, CA 94304
Tel: +1 650-846-1000
Toll Free: 1 800-420-8450
Fax: +1 650-846-1005

© 2006, TIBCO Software Inc.

All rights reserved. TIBCO, the TIBCO logo, The Power of Now, and TIBCO Software are trademarks or registered trademarks of TIBCO Software Inc. in the United States and/or other countries. All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

Table of Contents

Installation and Set Up	3
Project Migration Steps	3
Required Steps	3
Optional Steps	3
Project Migration for Firefox	4
Character Encoding	4
XPath and XSLT Requirements.....	4
Class Loading	5
The jsxlt Parameter.....	5
The jsx3.require() Method.....	6
Optional Legacy Classes	6
Optional Classes.....	7
Configuration Files	7
Paths	7
Internet Explorer Parameters.....	7
Custom Add-ins	8
Data Mapping	8
Updating Rules Files.....	8
Modifying the loadResource() Method Call	8
setOutboundStubURL() and setInboundURL() Methods.....	9
XSL Changes	9
Matrix Components	9
Relative Paths	9
Browsers and Layouts	10
Deployment Parameters	10
Logging File	11
Sound.....	11
Class Loading Options.....	11
Migrating Projects from 3.0 to 3.1.1	12
Class Hierarchy Related to jsx3.gui.BlockX.....	12
Checking for Equality against Model.getInstanceOf()	13
Model.findDescendants() Results Order.....	13
List.selectRecord() and Model Events	13

Installation and Set Up

Complete these steps before you migrate your projects:

1. Download and install TIBCO General Interface Builder.
2. Choose **GI_Builder.html** or **GI_Builder.xhtml** in the installation directory to start TIBCO General Interface Builder in Firefox or Internet Explorer.
Note: You can also choose to run in console mode, **GI_Builder.hta**. Console mode runs **only** in Internet Explorer.
3. After General Interface Builder initializes, choose or create a *workspace* directory. The workspace is the directory that contains your projects, custom add-ins, custom prototypes, and your user settings for General Interface Builder.
4. Copy projects that you would like to migrate to 3.2 from your previous JSXAPPS/ folder into your new *workspace/JSXAPPS/* folder.
Note: Make a back up of all projects you are migrating.
5. Copy any custom user prototypes from your *user/prototypes* directory to the *workspace/prototypes* directory.
6. Copy any custom add-ins to the *JSX/addins* or *workspace/addins* directory. Typically, add-ins to be used by a team of developers would be saved to the *JSX/addins* directory and posted by an administrator to a location accessible to the team. Add-ins for individual use can be saved to the *workspace/addins* directory.

Project Migration Steps

Complete the following steps to migrate your projects to 3.2. The steps fall into two categories: required and optional. If your application will be deployed on Firefox, also complete the steps in [Project Migration for Firefox](#). For examples of project migration, see the updated General Interface samples in the *workspace/JSXAPPS/samples* directory.

Required Steps

The following steps are required for updating your project:

1. If you're migrating projects from General Interface 3.0, you need to update your projects in 3.1.1 before migrating to 3.2. Complete the steps listed in [Migrating Projects from 3.0 to 3.1.1](#) before continuing to the next step.
2. Specify class loading options. See [Class Loading](#).
3. If your projects use add-ins, update for add-in changes:
 - Charting and custom add-ins – If your project uses Charting or custom add-ins, enable the add-in on the Add-ins page of the Project Settings dialog (Project > Project Settings).
 - Custom add-ins must be updated to work in General Interface 3.2. See [Configuration Files](#).
 - Mapping add-in – If your project includes data mapping, see [Data Mapping](#).
4. If your projects have rules files, update all rules files. See [Data Mapping](#).
5. If your projects use any custom XSL, you must update the XSL. See [XSL Changes](#).
6. If you created Matrix components in General Interface 3.2 Beta 2, see [Matrix Components](#).
7. For Firefox deployment, replace all deprecated List and Grid components with Matrix components. List and Grid are not supported in Firefox.

Optional Steps

The following steps are optional:

1. Manually merge your customized *logger.xml* files with the new logging system configuration file (*install_dir/logger.xml*). The 3.2 *logger.xml* file has new attributes and functionality, such as sound for the logging system and class loading options. See [Logging File](#).
2. Replace all deprecated List and Grid components with Matrix components. List and Grid are not supported in Firefox. **Required for Firefox.**

3. For consistent layout behavior in both Firefox and Internet Explorer using a Block, follow the recommendations specified in [Browsers and Layouts](#).
4. Change paths to relative URLs for more flexible development and deployment. See [Relative Paths](#).

Project Migration for Firefox

Character Encoding

If you're having trouble with files in Firefox, it could be due to an encoding problem. Projects that were localized prior to General Interface 3.2 may have been saved in UTF-16 encoding. If you open these files in General Interface 3.2, they might contain junk characters. Although General Interface 3.2 might read these Unicode files, it's best to resave the project in General Interface 3.2 on Internet Explorer before you proceed with the project migration. **Be sure to make a backup of your project before beginning.**

For applications loaded from the local disk, such as General Interface Builder, Firefox can only read non-XML files that are encoded in a standard 8-bit encoding. Firefox can read local XML files in any encoding supported by the host system only if the encoding is included in the XML declaration.

To re-encode any non-XML files from UTF-16 to 8-bit ASCII, you can use General Interface Builder 3.2 running in Internet Explorer. (**Note:** We do **not** recommend saving Unicode files in standard external editors, because results can be unpredictable.) In General Interface Builder, make sure the **Output character encoding** setting in the IDE Settings dialog (Tools > IDE Settings) is blank. Then open and re-save each file. You will not be able to include any non-ASCII characters in these plain text files. For the best compatibility with Firefox, all extended ASCII and 16-bit characters should be externalized in XML files that declare their character encoding in the XML declaration.

XML files do not need to be re-encoded as described above, although they can be. However, if an XML file is encoded in UTF-16 or any other non-ASCII character encoding, the encoding must be added to the XML declaration. Add or modify the first line of the XML file with the following XML declaration:

```
<?xml encoding="UTF-16"?>
```

Note that component serialization files are also XML files. If they have been encoded in UTF-16, they must also be modified as described above.

XPath and XSLT Requirements

XSL must meet these requirements to work properly in Firefox:

- The XSL must include the following namespace:
`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`
- The XSL must output valid XML. Balanced tags are required. For example,
`<xMessage>Hello World!</xMessage>`
 - The only output formats supported for XSLT processing are HTML 4.0 and XML.
 - XSLT implementation does not support the namespace axis, limiting the ability to query and discover namespaces. The DOM-based interface also fails to implement this axis.

- XSLT implementation does not support the `node-set()` method, which means that complex parameters and result tree fragments cannot be resolved.
- XSLT implementation does not allow output escaping to be disabled, which means that escaped entities cannot be resolved during a transformation.

Class Loading

General Interface now supports dynamic class loading for more efficient performance. Dynamic class loading, also known as lazy loading, means that classes are loaded as they're needed at the last possible moment.

These new class loading features have been introduced in 3.2:

- The `jsx1t` parameter
- The `jsx3.require()` method

The `jsx1t` Parameter

The `jsx1t` deployment parameter is a runtime configuration parameter that determines how classes are loaded. The `jsx1t` deployment parameter is located in the `script` tag on the web page that launches the deployed application. For example,

```
<script type="text/javascript" src="JSX/js/JSX30.js"
  jsxapppath=" ../TIBCO_GI/JSXAPPS/project_dir/"
  jsxmanualhome="true"
  jsx1t="true"
>
</script>
```

If you don't want to use dynamic loading for your 3.1 classes, set the `jsx1t` parameter to `false` or remove it. However, if you want to take advantage of dynamic class loading, you need to add the `jsx1t` parameter to your launch page or create a new launch page with the Deployment Utility.

When the `jsx1t` parameter is `true`, classes are loaded as follows:

1. All required classes are loaded as the system initializes.
2. Optional classes may be loaded during the execution of an application by:
 - A component file
The system class loader reads a component serialization file and automatically loads the appropriate classes.
 - The `jsx3.require()` method. See [The `jsx3.require\(\)` Method](#).

When the `jsx1t` parameter is `false`, the following occurs:

1. All required classes are loaded as the system initializes.
2. All General Interface 3.1 optional classes are loaded. No dynamic loading is used for the 3.1 classes. Note that General Interface 3.2 classes are always dynamically loaded regardless of this setting. This is for backward compatibility with General Interface 3.1.
3. During execution of an application, 3.2 optional classes are loaded by:
 - A component file
 - The `jsx3.require()` method. See [The `jsx3.require\(\)` Method](#).

The `jsx3.require()` Method

The `jsx3.require()` method can be used to load classes explicitly. Any code that references optional classes and executes before the class is loaded must use the `jsx3.require()` method to load the class. Use the fully qualified class name when using this method. For example,

```
jsx3.require("jsx3.net.Form");
```

Only classes that can be found by the system class loader are loaded. Custom classes can be added on the Classpath page of the Project Settings dialog (Project > Project Settings).

The `jsx3.require()` method must be called at least once before making these types of references:

- A static reference to `jsx3.gui.**`
- A reference to any optional classes that aren't subclasses of `jsx3.app.Mode1`, such as `jsx3.net.Form`

Optional Legacy Classes

These optional legacy classes are dynamically loaded in General Interface 3.2 when the script parameter `jsx1t` is set to `true`. Otherwise, they are loaded at start time.

```
jsx3.app.Monitor  
jsx3.app.UserSettings  
jsx3.gui.Alerts  
jsx3.gui.Block  
jsx3.gui.BlockX  
jsx3.gui.Button  
jsx3.gui.CheckBox  
jsx3.gui.Column  
jsx3.gui.DatePicker  
jsx3.gui.Dialog  
jsx3.gui.Form  
jsx3.gui.Grid  
jsx3.gui.Heavyweight  
jsx3.gui.HotKey  
jsx3.gui.ImageButton  
jsx3.gui.Interactive  
jsx3.gui.LayoutGrid  
jsx3.gui.List  
jsx3.gui.Menu  
jsx3.gui.Painted  
jsx3.gui.RadioButton  
jsx3.gui.Select  
jsx3.gui.Slider  
jsx3.gui.Sound  
jsx3.gui.Splitter  
jsx3.gui.Stack  
jsx3.gui.StackGroup  
jsx3.gui.Tab  
jsx3.gui.TabbedPane  
jsx3.gui.TextBox  
jsx3.gui.ToolBarButton  
jsx3.gui.Tree  
jsx3.gui.Window  
jsx3.gui.WindowBar  
jsx3.net.Form  
jsx3.xml.Cacheable
```

Optional Classes

These classes are dynamically loaded in General Interface 3.2, since they are not required and are from version 3.2 or later.

```
jsx3.gui.ColorPicker  
jsx3.gui.Image  
jsx3.gui.Matrix  
jsx3.gui.MatrixColumn  
jsx3.gui.TimePicker  
jsx3.net.Service  
jsx3.util.NumberFormat
```

Configuration Files

The application configuration file, `config.xml` (`workspace/JSXAPPS/project_dir/config.xml`), contains configuration parameters that affect deployment of the application.

Paths

Now that General Interface Builder 3.2 supports relative paths, many of the paths in the `config.xml` file are updated to relative paths. When you open a 3.1 project, General Interface Builder does this for you automatically after the upgrade prompt.

However, you must modify the path for the initial component. You can do this on the Deployment page of the Project Settings dialog (Project > Project Settings) in the IDE or in the configuration file. Simply remove `JSXAPPS/project_dir/` from the path as shown in Figure 2:

Figure 1: config.xml for 3.1

```
<record jsxid="objectseturl" type="string">JSXAPPS/project_dir/components/appCanvas.xml</record>
```

Figure 2: Revised config.xml for 3.2

```
<record jsxid="objectseturl" type="string">components/appCanvas.xml</record>
```

Internet Explorer Parameters

Remove the following Internet Explorer-specific parameters, as this functionality has been deprecated. However, you can pass these parameters at runtime using the General Interface runtime parameters. See [Deployment Parameters](#).

```
<record jsxid="xmlregkey" type="string">Msxml2.FreeThreadedDOMDocument.3.0</record>  
<record jsxid="xslregkey" type="string">Msxml2.XSLTemplate.3.0</record>  
<record jsxid="httpregkey" type="string">Msxml2.XMLHTTP</record>
```

Custom Add-ins

If you've created a custom add-in for General Interface 3.1, you need to add this new record to the project config.xml file (*workspace/JSXAPPS/project_dir/config.xml*):

```
<record jsxid="jsxversion" type="string">3.2</record>
```

You might also want to update your add-ins to use the new General Interface features, such as class loading and relative paths.

Data Mapping

For data mapping, there are two steps to updating your project:

1. Update rules files.
2. Modify the `loadResource()` method call in the JavaScript code.

For examples, see these samples in the *workspace/JSXAPPS/samples* directory — *WSDL_Mapping_1* and *WSDL_Mapping_2*.

Updating Rules Files

The format of data mapping rules files in 3.2 has changed. Files from 3.1 will not run in 3.2. General Interface Builder includes logic for converting 3.1 rules files to 3.2. Simply open each 3.1 rules file in the XML Mapping Utility in General Interface Builder 3.2 and save it. The 3.1 rules file will be updated automatically to the 3.2 format. To open the XML Mapping Utility, choose **Tools > Communication > XML Mapping Utility**.

Modifying the `loadResource()` Method Call

The JavaScript code generated by the XML Mapping Utility has changed due to a signature change in the `loadResource()` method (`jsx3.app.Server`). The rules file ID is now passed as a parameter instead of the URL. Update all legacy code generated by the XML Mapping Utility as shown in Figure 4.

Figure 3: Code for 3.1

```
var objService = new jsx3.net.Service(Rules_File_URL, Operation_Name);  
objService.setNamespace(namespace);
```

Figure 4: New Code for 3.2

```
var objService = Server_Name.loadResource(Project_Resource_File_Id);  
objService.setOperation(Operation_Name);
```

setOutboundStubURL() and setInboundURL() Methods

Note that these URLs are now resolved relative to the context server. For example, if the project directory for the context server is `test`, then the following inputs (all of which are valid) are equivalent:

```
jsxapp://test/xml/typical.xml  
xml/typical.xml  
JSXAPPS/test/xml/typical.xml
```

XSL Changes

The General Interface XSL templates for the classes that implement the `jsx3.xml.Cacheable` interface (Select, Menu, List, Grid, Matrix, and Tree) have changed in 3.2 and are not backwards compatible. Existing custom templates must be recreated starting from the default 3.2 XSL templates, which are located in the `JSX/xsl` directory. Because custom templates will not be supported in the future, this functionality is deprecated.

General Interface 3.2 introduces XML transformers as the preferred replacement for custom XSL templates. XML transformers are used to transform the source XML of a GUI control implementing the `jsx3.xml.Cacheable` interface before the XML is stored in the XML cache. For example, a transformer could transform non-CDF source XML into CDF-compliant XML or affect the visual style of the control by constructing a `@jsxstyle` CDF attribute from other information in the source document. For more information, see the API documentation for `jsx3.net.Cacheable` and the inline IDE documentation for the XML Transformers property in the Property Editor palette.

For Firefox, XSL must meet certain requirements. See [Project Migration for Firefox](#).

Matrix Components

Matrix components created in the General Interface 3.2 Beta 2 cause “no such method” or “function required” exceptions when loaded with General Interface 3.2. This is due to obfuscated variants in the serialization files. Check any serialization file created in General Interface Builder 3.2 Beta 2 which contains a `jsx3.gui.Matrix` for obfuscated variant attributes, such as `hk=""`. Remove these obfuscated variants manually in Source XML (Expert) view in General Interface Builder or use a text edit tool to search and remove all instances. After resaving the serialization file, open the file in General Interface Builder 3.2 and validate it.

Relative Paths

Now that General Interface Builder supports relative paths, you can update your project files to use relative paths. Although this isn't required, specifying relative paths to application resources increases the portability of code from one project to another and prevents problems when renaming projects.

Modify paths in the following files:

- Modify the path for the initial component in the Project Settings dialog or the configuration file. See [Configuration Files](#).

- Modify paths in JavaScript code to use relative URLs. To update your code, remove this portion of the path: `JSXAPPS/project_dir/`.
- Modify paths in the component serialization files or in the Property Editor palette to use relative URLs.

For more information, see URI Resolution in TIBCO General Interface Builder Getting Started Guide and the General Interface API documentation.

Browsers and Layouts

To avoid unexpected layout behavior when using relative positioning, such as misaligned GUI components, it's recommended to use Block as a container **only** if it meets at least **one** of these requirements:

- The Block is owned by a layout manager, such as LayoutGrid, Tab, Stack, and Splitter.
- The Block is relatively positioned and has a width of 100%.
- The Block is absolutely positioned.

Deployment Parameters

The HTML `script` element has several new parameters that are significant in the General Interface runtime:

- `jsxappns` - Application namespace. For example, `jsxappns="myAPP"`. Since every application deployed on a single page must have a unique namespace, overriding the namespace can be useful when an application is multi-instantiated on the same page. However, the application must be written to never reference its namespace directly.
- `jsxapploader` - Specifies what type of progress bar to load when the application launches. For example, `jsxapploader="0"`.
 - 0 = standard progress bar
 - 1 = portlet, subordinate progress bar
- `jsxapppath` - The path to the application. For example, `jsxapppath=" ../workspace/JSXAPPS/samples/chart"`.

The other mechanism for configuring an application is to modify the application configuration file, `project_dir/config.xml`. This file contains configuration parameters that affect deployment of the application. These parameters can be edited on the Deployment Page of the Project Settings dialog in TIBCO General Interface Builder. To open the Deployment Page, choose **Project > Project Settings > Deployment**.

The General Interface runtime or JSX system, which is required to run deployed General Interface applications, is configured at deployment with the same `script` element mechanism described before.

- `jsxlt` - Determines how classes are loaded. See [The jsxlt Parameter](#).
- `jsx_logger_config` - Specifies the name and location of the logging system configuration file to be used.
- The following Internet Explorer-specific parameters:
 - `jsxxmlregkey` - The default ActiveX XML object key. Specifies the version of the XML Parser instance as referenced in the Microsoft Windows Registry. For example, `jsxxmlregkey="Msxm12.FreeThreadedDOMDocument.4.0"`.
 - `jsxxs1regkey` - The default ActiveX XSL object key. Specifies the version of the XSL Parser instance as referenced in the Microsoft Windows Registry. For example, `jsxxs1regkey="Msxm12.XSLTemplate.4.0"`.

- `jsxhttpregkey` - The default ActiveX HTTP object key. Specifies the version of the HTTP control as referenced in the Microsoft Windows Registry. For example, `httpregkey="Msxml2.XMLHTTP.4.0"`.
- `jsxxmlversion` - Specifies the version of the previous three keys — XML Parser, XSL Parser, and the HTTP control, such as version 3, 4, or 5. For example, `jsxxmlversion="3"`.

By default, General Interface loads version 4 if available. Otherwise, version 3 is loaded. If you need to use a different version, use these system parameters to override the defaults.

Logging File

The logging system configuration file, `GI_HOME/logger.xml`, has several new features:

- Sound for the logging system
- Class loading options

For more information on logging, see Logging and Application Monitors in TIBCO General Interface Builder Getting Started Guide.

Sound

To enable sound for error and warning messages,

1. Open `logger.xml`, located in the `GI_HOME` directory.
2. Change the `beepLevel` property for the `ide` handler from `jsx3.util.Logger.OFF` to a logging level, such as `jsx3.util.Logger.ERROR`. Sound is played for the specified logging level and higher.

For example, if the `beepLevel` property is `jsx3.util.Logger.ERROR`, a sound plays whenever error and fatal messages are reported.

Important: Firefox requires a plug-in that can play `.wav` files in order to play the sounds.

Class Loading Options

Handlers have two new attributes that affect class loading:

- `lazy` — If true, the logging system waits for the class to load on its own. Use `lazy` or `require` but not both.
- `require` — If true, the class is loaded immediately using the synchronous class loading mechanism. If the application is set to run with dynamic class loading (`jsx1t`) and the handler `class` attribute is an optional class (for example, `jsx3.app.Monitor`), this must be set to true. Use `lazy` or `require` but not both.

Migrating Projects from 3.0 to 3.1.1

Before migrating projects to 3.2, you need to update projects in 3.1.1 as follows:

1. Download General Interface 3.1.1 from the TIBCO Developer Network at <http://developer.tibco.com> and install.
2. Make backup copies of your projects before migrating.
3. Update all component files using one of these methods:
 - Open all projects in 3.1.1, open all component files, and save. This automatically converts component files to the new GUI package structure and updates the serialization files.
 - Manually edit the package name for all components. For example, change `jsx3.TextBox` to `jsx3.gui.TextBox`.
 - Download the add-in from the TIBCO Developer Network at <http://developer.tibco.com>, when available. The add-in automatically converts component files to the new GUI package structure and updates the serialization files.
4. Update mapping rules files to the 3.1.1 format. Simply open each rules file in the XML Mapping Utility in General Interface Builder 3.1.1 and save it. The rules file will be updated automatically to the 3.1.1 format. To open the XML Mapping Utility, choose **Tools > Communication > XML Mapping Utility**. You'll also need to do this in General Interface Builder 3.2.
5. Modify JavaScript code due to changes in 3.1. Read the following topics to see if you need to modify your code:
 - [Class Hierarchy Related to `jsx3.gui.BlockX`](#)
 - [Checking for Equality against `Model.getInstanceOf\(\)`](#)
 - [`Model.findDescendants\(\)` Results Order](#)
 - [`List.selectRecord\(\)` and Model Events](#)
6. Migrate the updated projects to 3.2 as described in [Project Migration Steps](#).

Class Hierarchy Related to `jsx3.gui.BlockX`

In General Interface 3.0, `jsx3.List`, `jsx3.Select`, `jsx3.Tree`, and `jsx3.chart.ChartComponent` all extended `jsx3.BlockX`. These classes inherited the methods for storing XML and XSL data in the application cache from `BlockX`.

In 3.1, these methods have been moved to the mixin interface `jsx3.xml.Cacheable`. Therefore, `jsx3.gui.List`, `jsx3.gui.Select`, `jsx3.gui.Tree`, and `jsx3.chart.ChartComponent` no longer extend `jsx3.gui.BlockX`. Any code that relied on this specific 3.0 class hierarchy will not work in 3.2. For example, the following code will **not** work:

```
function alertIfCDF(objControl) {
    if (objControl instanceof("jsx3.gui.BlockX"))
        objControl.getServer().alert("Alert", objControl.getName() + " is a CDF control");
}
```

It should be changed to:

```
function alertIfCDF(objControl) {
    if (objControl instanceof(javax.xml.CDF))
        objControl.getServer().alert("Alert", objControl.getName() + " is a CDF control");
}
```

The method `isSubclassOf()` is similarly affected.

Checking for Equality against Model.getInstanceOf()

The method `jsx3.Model.getInstanceOf()` has been deprecated in 3.1. (The Class Inheritance and Introspection document describes the preferred methods for determining whether an object is an instance of a class or interface.) Because of the package reorganization in 3.1, `getInstanceOf()` does not always return the same value as in 3.0. For example, if `getInstanceOf()` returned `jsx3.Block` in 3.0, it will return `jsx3.gui.Block` in 3.1. Therefore any code that checks the return value of `getInstanceOf()` for equality will likely break in 3.1. For example:

```
// this will break in 3.1
if (objBlock.getInstanceOf() == "jsx3.Block")
    objBlock.getServer().alert("Alert", "It's a block!");
```

Model.findDescendants() Results Order

General Interface 3.0 included a bug in the method `jsx3.Model.findDescendants()` that caused the results to be returned in reverse order. This was in violation of the method contract that said the results would be returned in either depth-first or breadth-first order. This bug is fixed in 3.1. However, any code that relied on the order of the results from this method may break in 3.1.

The following methods that call `findDescendants()` are also affected: `Model.getDescendantOfName()`, `Model.getFirstChildOfType()`, and `Model.getDescendantsOfType()`. The method `isSubclassOf()` would be similarly affected.

List.selectRecord() and Model Events

The contract of the method `jsx3.gui.List.selectRecord()` has changed. In 3.0 it caused the SELECT model event to fire. In 3.1 it never causes the model event to fire. This change is related to the new 3.1 model event protocol detailed in Model Events document. For an application running under the 3.0 model event protocol to continue to function properly in 3.1, `List.selectRecord()` should be replaced with `List.doSelect()`. Otherwise, the application must be upgraded to the 3.1 model event protocol. All other methods affected by the new event protocol are backwards compatible in 3.1.