

# **TIBCO® General Interface - Enterprise Edition**

## **Migration Guide**

*Software Release 3.9.1  
April 2012*

## Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN TIBCO GENERAL INTERFACE INSTALLATION) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIB, TIBCO, TIBCO Adapter, Predictive Business, Information Bus, The Power of Now, TIBCO General Interface, TIBCO General Interface Framework, TIBCO General Interface Builder, TIBCO General Interface Performance Profiler, and TIBCO General Interface Test Automation Kit are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

EJB, Java EE, J2EE, and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and 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.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README.TXT FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

U.S. Patent No. 8,136,109

Copyright © 2001-2012 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

# General Interface Migration

General Interface Migration

Software Release 3.9

March 3.9

- [Chapter 1 General Interface Migration Paths](#)
- [Chapter 2 Migrating Projects from 3.0 to 3.4](#)
- [Chapter 3 Migrating Projects from 3.1.x to 3.4](#)
- [Chapter 4 Migrating Projects from 3.2 or 3.3 to 3.4](#)
- [Chapter 5 Migrating Projects from 3.x to 3.5](#)
- [Chapter 6 Migrating Projects from 3.x to 3.6-3.8](#)

# Chapter 1 General Interface Migration Paths

This manual explains how to migrate projects from previous General Interface releases to the current release.

The migrations paths are as follows:

- General Interface 3.0 to 3.4  
The migration path is 3.0 > 3.1.x > 3.2 or 3.3 > 3.4. See [Migrating Projects from 3.0 to 3.4](#).
- General Interface 3.1.x to 3.4  
The migration path is a direct migration from 3.1.x > 3.2 or 3.3 > 3.4. See [Migrating Projects from 3.1.x to 3.4](#).
- General Interface 3.2 or 3.3 to 3.4  
The migration path is 3.2 or 3.3 > 3.4. See [Migrating Projects from 3.2 or 3.3 to 3.4](#).
- General Interface 3.4 to 3.5.  
No migration is required from General Interface 3.4 to 3.5.  
For more information on General Interface, see General Interface Release Notes.
- General Interface 3.4 or 3.5 to version 3.6-3.8.  
No migration is required from General Interface 3.4 or 3.5 to version 3.6-3.8, but there is a change in behavior from previous version.  
See [Migrating Projects to 3.6 or 3.7](#) for information about how to modify your application if it uses the <resourcename>.meta.xml file.

## Chapter 2 Migrating Projects from 3.0 to 3.4

This chapter discusses how to migrate General Interface 3.0 - Enterprise Edition projects to 3.4. The migration path is 3.0 > 3.1.x > 3.2 or 3.3 > 3.4. No migration is required from 3.2 to 3.3.

- [Installation and Set Up for 3.0 to 3.4 Migration](#)
- [Required Steps for 3.0 to 3.4 Migration](#)

### Installation and Set Up for 3.0 to 3.4 Migration

Before you migrate your General Interface 3.0 - Enterprise Edition projects to 3.4, you must migrate to 3.1.x and 3.3. First, complete these steps to migrate your 3.0 projects to 3.1.x:

1. Install General Interface 3.1.x - Enterprise Edition.
2. Make backup copies of your projects before migrating.
3. Copy projects that you would like to migrate from the 3.0 JSXAPPS folder into the 3.1.x JSXAPPS folder.
4. Double-click `GI_Builder.html` in the 3.1.x installation directory to start General Interface Builder.

### Required Steps for 3.0 to 3.4 Migration

Complete the following steps to migrate your General Interface 3.0 projects to 3.1.x. After you migrate to 3.1.x, you need to migrate to 3.3 and 3.4.

1. Open all projects and project files in General Interface Builder 3.1.x to update all component files to the new 3.1.x package structure. Component files are automatically converted to the new 3.1.x GUI package structure and the serialization files are also updated.
2. Modify all class names and constructors in your JavaScript logic files to the new 3.1.x package structure. For more information, see [Package Reorganization in General Interface Release Notes](#).
3. Modify JavaScript code due to changes in 3.1.x. 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](#)
4. If your projects use data mapping, update the project as described in [Data Mapping for 3.0 to 3.4 Migration](#).
5. Migrate the updated 3.1.x projects to 3.3 and 3.4 as described in [Migrating Projects from 3.1.x to 3.4](#).

### Data Mapping for 3.0 to 3.4 Migration

This section explains the steps for migrating General Interface 3.0 projects that use data mapping to General Interface 3.1.x.

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

1. Update rules files.
2. Update JavaScript code.

### Updating Rules Files

Update mapping rules files to the 3.1.x format. Simply open each rules file in the XML Mapping Utility (formerly SOAP Mapping Utility) in General Interface Builder 3.1.x and save. The rules file is automatically updated to the 3.1.x 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 or 3.3.

### Updating JavaScript Code

There are two steps for updating JavaScript code to 3.1.x:

1. Update the function code that calls the service.
2. Update method calls that are associated with GUI objects.

### Updating the Function Code

The code for calling the `jsx3.Service` has changed and needs to be updated to use the new `jsx3.net.Service`.

To update the function code that invokes the service, complete these steps:

1. Open the mapping rules file in General Interface 3.1.x.
2. Click the Generate button on the XML Mapping Utility toolbar and choose the operation from the drop-down list.
3. Replace the 3.0 function code in the JavaScript logic file with the new 3.1.x function code.

### Updating Method Calls

You also need to replace any method calls in your JavaScript code that are associated with GUI objects with the new method calls.

### 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.x, 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 3.3. 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 jsx3.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 `jsx3.Model.getInstanceOf()` method 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, the `getInstanceOf()` method 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 defect in the `jsx3.Model.findDescendants()` method 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 defect 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 `isSubclassOf()` method would be similarly affected.

## `List.selectRecord()` and Model Events

The contract of the `jsx3.gui.List.selectRecord()` method has changed. In 3.0, it caused the SELECT model event to fire. In 3.1, it never caused the model event to fire. This change is related to the new 3.1 model event protocol detailed in the 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.

## Chapter 3 Migrating Projects from 3.1.x to 3.4

This chapter explains how to migrate your General Interface 3.1.x - Enterprise Edition projects to 3.4. The migration path is 3.1.x > 3.2 or 3.3 > 3.4. No migration is needed from 3.2 to 3.3.

The migration steps fall into two categories: required and optional. If your application will be deployed on Firefox, also complete the steps in [Migrating Projects for Firefox for 3.1.x to 3.4](#).

- [Browsers and Layouts for 3.1.x to 3.4 Migration](#)
- [Class Loading in Migration for 3.1.x to 3.4 Migration](#)
- [Data Mapping for 3.1.x to 3.4 Migration](#)
- [Installation and Set Up for 3.1.x to 3.4 Migration](#)
- [Migrating Projects for Firefox for 3.1.x to 3.4](#)
- [Optional Migration Steps for 3.1.x to 3.4 Migration](#)
- [Project Settings for 3.1.x to 3.4 Migration](#)
- [Relative Paths for 3.1.x to 3.4 Migration](#)
- [Required Migration Steps for 3.1.x to 3.4 Migration](#)
- [Required Steps for Firefox for 3.1.x to 3.4 Migration](#)
- [XSL Changes for 3.1.x to 3.4 Migration](#)

### Browsers and Layouts for 3.1.x to 3.4 Migration

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.

### Class Loading in Migration for 3.1.x to 3.4 Migration

General Interface 3.3 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.

#### jspxlt Parameter



The `jspxlt` parameter is no longer supported in 3.4. If you're migrating to 3.4, skip this topic.

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



```

<script type="text/javascript" src="JSX/js/JSX30.js"
  jsxapppath=" ../workspace/JSXAPPS/PROJECT_DIR/"
  jsxmanualhome="true"
  jsx1t="true"
>
</script>

```

When the `jsx1t` deployment parameter is set to `true`, the default setting, all required classes are loaded as the system initializes. Optional classes are loaded by the component file and the `jsx3.require()` method.

If you don't want to use dynamic loading for your 3.1.x classes, set the `jsx1t` parameter to `false` (`jsx1t="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 General Interface 3.3 Deployment Utility (Project > Deployment Utility).

### jsx3.require() Method

The `jsx3.require()` method can be used to load classes explicitly. Use the fully qualified class name when using the `jsx3.require()` 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 panel of the Project Settings dialog (formerly Deployment Options). To open the Project Settings dialog, choose **Project > Project Settings**.

When a component file is deserialized, the class of each object encountered in the file is dynamically loaded if it's not already loaded. Therefore, it's often not necessary to use the `jsx3.require()` method with classes that descend from `jsx3.app.Model`. However, if JavaScript code references these classes and if the code executes before a deserialization automatically loads the class, you must use the `jsx3.require()` method to explicitly load the class.

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

- A static reference to a class descending from `jsx3.gui.Model` (typically `jsx3.gui.**`).
- Any references to subclasses of `Model` that execute before the class is loaded through object deserialization.



The General Interface Builder debugger classes are dynamically loaded. To use the JavaScript Step Through Debugger in General Interface Builder, you must use the `jsx3.require()` method before any `jsx3.ide.debug()` statements to load debugger classes. For example, `jsx3.require("jsx3.ide.Debugger");`

## Data Mapping for 3.1.x to 3.4 Migration

This section explains the steps for migrating General Interface 3.1.x projects that use data mapping to General Interface 3.2 and 3.3.

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.

## Updating Rules Files

The format of data mapping rules files has changed in General Interface releases after 3.1.x. Mapping rules files from 3.1.x will not run in 3.3. General Interface Builder includes logic for converting 3.1.x rules files to 3.3. Simply open each 3.1.x rules file in the XML Mapping Utility (formerly SOAP Mapping Utility) in General Interface Builder 3.3 and save it. The 3.1.x rules file is automatically updated to the 3.3 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 the [New Code for 3.3](#), below.

### Code for 3.1.x

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

### New Code for 3.3

```
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 are valid and equivalent:

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

If the `test` project directory is nested in a subdirectory, such as `JSXAPPS/samples/test`, then the following inputs are valid and equivalent: `jsxapp://samples/test/xml/typical.xml`

```
samples/test/xml/typical.xml
JSXAPPS/samples/test/xml/typical.xml
```

## Installation and Set Up for 3.1.x to 3.4 Migration

Before you can migrate your General Interface 3.1.x projects to 3.4, you need to migrate to 3.3. Complete these steps to migrate your 3.1.x projects to 3.3:

1. Install General Interface 3.3. For more information about installing General Interface, see [General Interface Installation](#).
2. Double-click `GI_Builder.html` or `GI_Builder.xhtml` in the installation directory to start General Interface Builder in Firefox or Internet Explorer.
3. Choose or create a workspace directory after General Interface Builder initializes. The

workspace is the directory that contains your projects, custom add-ins, custom prototypes, and your user settings for General Interface Builder. Separation of the General Interface install directory from the workspace allows for easier application deployment. For more on workspaces, see the General Interface Developer Guide.

4. Make backup copies of all projects you are migrating.
5. Copy projects that you would like to migrate to 3.3 from your previous JSXAPPS folder into your new workspace/JSXAPPS folder.
6. Copy any custom user prototypes from your user/prototypes directory to the workspace /prototypes directory. Components saved to this folder display in the User folder of the Component Libraries palette.
7. 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.

## Migrating Projects for Firefox for 3.1.x to 3.4

This section discusses how to migrate projects to be deployed on Firefox from 3.1.x to 3.3.

### GUI Components

Replace all deprecated List and Grid components with Matrix components. List and Grid are not supported in Firefox.

### Character Encoding

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 3.3, they might contain junk characters. Although General Interface 3.3 might read these Unicode files, it's best to resave the project in General Interface Builder 3.3 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 reads only 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 3.3 running in Internet Explorer or a text editor.

To re-encode non-XML files from UTF-16 to 8-bit ASCII, complete these steps:

1. Open General Interface Builder in Internet Explorer.
2. Choose **Tools > IDE Settings** to open the IDE Settings dialog.
3. Make sure the **Output character encoding** field is blank.
4. 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.

## Optional Migration Steps for 3.1.x to 3.4 Migration

The following steps are optional:

1. Manually merge your customized `logger.xml` files with the new logging system configuration file (`GI_HOME/logger.xml`).  
In General Interface 3.2 and later releases, the `logger.xml` file has new attributes and functionality, such as sound for the logging system and class loading options. The logging file has also been moved from the `GI_HOME/JSX` directory to `GI_HOME` and must be deployed accordingly. For more information, see the General Interface Developer Guide.
2. To take advantage of MSXML 4 or later, if installed, remove the Internet Explorer parameters from the application configuration file. See [Internet Explorer Parameters](#).
3. For consistent layout behavior in both Firefox and Internet Explorer using a Block, follow the recommendations specified in [Browsers and Layouts for 3.1.x to 3.4 Migration](#).
4. Change paths to relative URLs for more flexible development and deployment. See [Relative Paths for 3.1.x to 3.4 Migration](#).
5. Replace all deprecated List and Grid components with Matrix components. Note that List and Grid are not supported in Firefox.
6. Rename resource files from `.jss` to `.xml` for easier deployment. For example, Microsoft IIS6 Server doesn't handle the `.jss` file extension correctly. Renaming resource files with

the .xml extension prevents such deployment problems.

## Project Settings for 3.1.x to 3.4 Migration

When you create a new project in General Interface Builder, a default application configuration file is automatically created as part of the project in the project directory: `workspace/JSXAPPS/project_dir_/config.xml`. The application configuration file contains application configuration data, such as project settings, application deployment, and file locations.

You can modify the project settings in the Project Settings dialog (Project > Project Settings) or in the application configuration file. Any changes you make in the Project Settings dialog are saved to the configuration file. Some changes can only be made in the configuration file.

### Paths

Application resources can now be specified using relative paths, which allows for easier portability of code from one project to another and simplifies the relocation of applications in the JSXAPPS folder hierarchy.

When you open a 3.1.x project in General Interface Builder 3.3, paths in the `config.xml` file are updated automatically to relative paths after the upgrade prompt.

However, you must modify the path for the initial component. You can do this on the Deployment panel of the Project Settings dialog (Project > Project Settings) in the IDE or in the application configuration file (`workspace/JSXAPPS/PROJECT_DIR_/config.xml`). Simply remove `JSXAPPS/PROJECT_DIR_/` from the path as shown in the [Revised config.xml for 3.3](#) below.

### config.xml for 3.1.x

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

### Revised config.xml for 3.3

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

### Auto Load Options

New file auto load options have been introduced in General Interface 3.2. In prior General Interface releases, auto loading could be set to true or false. In General Interface 3.2 and above, there are four auto load options: Manually as needed, At init, At full init, and At light init. Available options vary by file type. For more information about auto load options, see the General Interface Developer Guide.

Before using the new options, you need to modify the `onLoad` `jsxid` for some files in the application configuration file as follows:

1. Open the application configuration file, which is located in `workspace/JSXAPPS/PROJECT_DIR_/config.xml`.
2. Find `jsxid="includes"` and notice that there are multiple records of type `map`. Each type `map` record has a record with `jsxid="onLoad"`.

3. Modify each child record of type map with a jsxid of onLoad as follows:
  - a. Change the jsxid onLoad value to load.
  - b. Change the type from boolean to number.
  - c. Change the record value to the new desired value, such as 0, 1, 2, or 3. For values, see the next table.

For example, to set logic.js to load automatically when the application initializes, change the record from this:

```
<record jsxid="onLoad" type="boolean">true</record>
```

To this:

```
<record jsxid="load" type="number">1</record>
```

Auto Load Option	3.2 or 3.3	Prior to 3.2
Manually as needed	0	false
At init	1	true
At full init	2	Not available
At light init	3	Not available

The Auto Load option is disabled for GUI component files, such as appCanvas.xml. You can specify a GUI component file to automatically load when the application initializes in the Initial Component field on the Deployment panel of the Project Settings dialog.

1. Save the configuration file and reload the project.

Once you've modified the application configuration file, you can also set auto load options in General Interface Builder. Right-click a file in the Project Files palette and choose Edit Profile. Select an option from the Auto Load drop-down list in the Edit Profile dialog and click Save.

## Internet Explorer Parameters

To take advantage of MSXML 4 or later, if installed, remove the following Internet Explorer parameters from the application configuration file.

1. Open the application configuration file: workspace/JSXAPPS/PROJECT\_DIR\_/config.xml.
2. Remove the following Internet Explorer-specific 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>
```

However, you can pass these parameters at runtime using the General Interface runtime parameters. See the General Interface Developer Guide.

## Custom Add-ins

If you've created a custom add-in for General Interface 3.1.x, you need to edit the project config.xml file of the add-in as follows:

1. Open the project `config.xml` file of the add-in located at workspace `/JSXAPPS/PROJECT_DIR_/config.xml`.
2. Add this new record to the configuration file:

```
<record jsxid="jsxversion" type="string">3.3</record>
```

3. Save the file.

You might also want to update your add-ins to use the new General Interface features, such as class loading and relative paths. See [Class Loading in Migration for 3.1.x to 3.4 Migration](#) and [Relative Paths for 3.1.x to 3.4 Migration](#).

## Relative Paths for 3.1.x to 3.4 Migration

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 (Project > Project Settings) or the application configuration file. See [Project Settings for 3.1.x to 3.4 Migration](#).
- 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 Properties Editor palette to use relative URLs.

For more information, see the General Interface Developer Guide.

## Required Migration Steps for 3.1.x to 3.4 Migration

The following steps are required for updating your 3.1.x project to 3.3:

1. Open the 3.1.x project and project files in General Interface Builder 3.3 and resave all project files.



If you're using Firefox and have files encoded in UTF-16, open the 3.1.x project and project files in Internet Explorer first and resave the files before opening in Firefox. See [Character Encoding](#).

2. Specify class loading options to improve application performance with the new dynamic class loading features in General Interface Builder 3.3. See [Class Loading in Migration for 3.1.x to 3.4 Migration](#).
3. Modify records that specify auto loading in the application configuration file. New auto load options have been added. See [Auto Load Options](#).
4. If your projects use add-ins, update the add-ins as follows:
  - Charting and custom add-ins If your project uses Charting or custom add-ins,

- enable the add-in on the Add-ins panel of the Project Settings dialog (Project > Project Settings).
  - Project settings Application configuration files must be modified for custom add-ins to work in General Interface 3.3. See [Project Settings for 3.1.x to 3.4 Migration](#).
  - Mapping add-in If your project includes data mapping, see [Data Mapping for 3.1.x to 3.4 Migration](#).
5. If your projects have rules files, update all rules files. See [Data Mapping for 3.1.x to 3.4 Migration](#).
  6. If your projects use any custom XSL, you must update the XSL. See [XSL Changes for 3.1.x to 3.4 Migration](#).



If any project files in your migrated project are displayed in red in the Project Files palette, you need to update the references to those files. Right-click the file in the Project Files palette and choose **Edit Profile**. Modify the path in the URI field and click **Save**.

7. Migrate your updated 3.3 projects to 3.4. See [Migrating Projects from 3.2 or 3.3 to 3.4](#).

## Required Steps for Firefox for 3.1.x to 3.4 Migration

The following additional steps are required for updating your 3.1.x project to 3.3 for Firefox deployment:

1. Open any files that are saved in UTF-16 encoding in Internet Explorer first and resave the files before opening in Firefox. See [Character Encoding](#).
2. Complete the required steps described in [Required Migration Steps for 3.1.x to 3.4 Migration](#).
3. Replace all deprecated List and Grid components with Matrix components. List and Grid are not supported in Firefox.
4. If you're using XSL files, verify that they meet certain requirements to work correctly in Firefox. See [XPath and XSLT Requirements](#).

There are also additional optional steps you can complete. See [Optional Migration Steps for 3.1.x to 3.4 Migration](#).

## XSL Changes for 3.1.x to 3.4 Migration

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 later releases and are not backwards compatible. Existing custom templates must be recreated starting from the default 3.3 XSL templates, which are located in the `GI_HOME/JSX/xsl` directory. Because custom templates will not be supported in the future, this functionality is deprecated.

Introduced in General Interface 3.2, XML transformers are 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 `jsx3.net.Cacheable` in General Interface API Reference and the inline IDE documentation for the XML Transformers property in the Properties Editor palette.

For Firefox, XSL must meet certain requirements. See [XPath and XSLT Requirements](#).

## Chapter 4 Migrating Projects from 3.2 or 3.3 to 3.4

This chapter explains how to migrate your 3.2 and 3.3 projects to General Interface 3.4. The migration path is 3.2 or 3.3 > 3.4. No migration is needed from 3.2 to 3.3.

- [Installation and Set Up to Migrate from 3.2 or 3.3 to 3.4](#)
- [Migration Steps to Migrate from 3.2 or 3.3 to 3.4](#)

### Installation and Set Up to Migrate from 3.2 or 3.3 to 3.4

Complete these steps to migrate your General Interface 3.2 or 3.3 projects to 3.4:

1. Make backup copies of all projects you are migrating.
2. Install General Interface 3.4. For more information about installing General Interface, see the General Interface Installation Guide.
3. Double-click `GI_Builder.html` or `GI_Builder.xhtml` in the installation directory to start General Interface Builder in Firefox or Internet Explorer.
4. Choose or create a workspace directory after General Interface Builder initializes. For more information on workspaces, see the General Interface Developer Guide.



Create a new workspace to get the updated 3.4 sample applications. Choosing an existing workspace doesn't replace previous sample applications with updated 3.4 sample applications. This built-in functionality is designed to prevent workspace files from getting overwritten.

5. If you chose an existing workspace, copy any custom add-ins to the 3.4 `JSX/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.
6. If you created a new workspace, complete these steps:
  - a. Copy projects that you would like to migrate to 3.4 from your previous workspace `/JSXAPPS` folder into your new workspace `/JSXAPPS` folder.
  - b. Copy any custom user prototypes from your workspace `/prototypes` directory to the new workspace `/prototypes` directory. Components saved to this folder display in the User folder of the Component Libraries palette.
  - c. Copy any custom add-ins to the 3.4 `JSX/addins` or new 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.

### Migration Steps to Migrate from 3.2 or 3.3 to 3.4

The following steps are required for updating your 3.2 and 3.3 project to 3.4:

1. Update the project for changes to class loading.

2. Review [Changes in Behavior in 3.4](#) to see if your applications are affected by changes to General Interface 3.4.
3. If your application creates a LayoutGrid - Side/Side component programmatically using the constructor instead of being deserialized from an XML file, call the `setCols()` method and pass an asterisk as the parameter to match the 3.3 behavior: `setCols("*")`. Otherwise, an extra row will be added to the layout. Note that the `setDimensionArray()` method has been deprecated in 3.4 and is replaced with the `setCols()` and `setRows()` methods.
4. Optional Rewrite any JavaScript code that uses deprecated APIs. See [Deprecated APIs in General Interface Release Notes](#).
5. Optional Replace any Block components that use an `iframe` with the new `IFrame` component.

## Class Loading

There are two changes to class loading in 3.4:

- The `At Lt Init` and `At Full Init` Auto Load options are no longer supported. If your project uses these class loading options, see [Auto Load Options](#).
- The `jsxlt` parameter is no longer supported. If your deployed projects use `jsxlt="false"`, see [jsxlt Parameter](#). If your project uses `jsxlt="true"`, no changes are needed.

## Auto Load Options

The Auto Load options for JavaScript files, `At Lt Init` and `At Full Init`, are no longer supported. If you've used these settings in your project, you need to reset the JavaScript file to Auto Load (`At Init`).

To change the Auto Load setting to `At Init`, complete these steps:

1. Right-click the JavaScript file in the Project Files palette.
2. Choose Auto Load from the context menu. The file name now displays in a bold font in the Project Files palette.

## jsxlt Parameter

The `jsxlt` launch parameter is no longer available as of 3.4. All applications now load in the `jsxlt="true"` mode. If you deployed your project using `jsxlt="false"`, use `jsx3.require()` to explicitly load classes, so that the project is compatible with dynamic class loading. See [jsx3.require\(\) Method](#).

## Changes in Behavior in 3.4

To see if any of these changes affect your applications, run your applications in General Interface 3.4 and check the following behaviors. Then modify your applications as needed.

- Menu
  - Menu positioning and cascading may be different. See [GI-63](#) in [General Interface Release Notes](#).
  - Long menus now render with auto-scroll arrows instead of with scroll bars.
- TabbedPane
  - Provides an automatic right and left scrolling mechanism for navigating through tabs that are hidden when they exceed the viewable space.

- Tree
  - Support for range selection using Shift+click and multiple selection using Ctrl+click.
  - Support for drag-and-drop of multiple records.
  - Drag-and-drop insertion between nodes and insertion as last child of node.
  - Context menu can operate on multiple rows.
- Dialog
  - When deserialized a dialog box top-left position could be at a negative position if the min-width and min-height were larger than the client window and no top-left positions were defined. Left and top are now never less than zero. However, since the resize control is in the bottom right, call `constrainPosition(true)` if there is a chance that the dialog is larger than its parent.
- Form elements
  - All form elements now inherit font color from an ancestor block.
- `setValue()` and Matrix
  - If a Matrix is a single selection and the passed value is an array with more than one element, an illegal argument exception is now thrown.
- `jsx3.app.Model.getChild()` returns null instead of undefined. Unless specifically documented, application should be careful about checking against undefined or null value returned by General Interface functions. Developer should check against known value.

For example, instead of this:

```
if (objJSX.getChild('block') == null)
```

or

```
if ( typeof(objJSX.getChild('block')) == 'undefined' )
```

Do this instead:

```
if ( !(objJSX.getChild('block') instanceof jsx3.app.Model) )
```

or

```
if ( ! (objJSX.getChild('block') )
```

For more information on new features, changes in functionality, and known and closed issues, see General Interface Release Notes.

## Chapter 5 Migrating Projects from 3.x to 3.5

Migrating from General Interface 3.4 to 3.5 requires no migration—simply open version 3.4 projects in 3.5.

- To migrate from 3.0 to 3.5, follow the instructions in [Migrating Projects from 3.0 to 3.4](#)
- To migrate from 3.1 to 3.5, follow the instructions in [Migrating Projects from 3.1.x to 3.4](#).
- To migrate from 3.2 or 3.3 to 3.5, follow the instructions in [Migrating Projects from 3.2 or 3.3 to 3.4](#)

## Chapter 6 Migrating Projects from 3.x to 3.6-3.8

This chapter describes how to migrate General Interface projects to version 3.6-3.8.

- To migrate from 3.0 to 3.6-3.8, follow the instructions in [Migrating Projects from 3.0 to 3.4](#).
- To migrate from 3.1 to 3.6-3.8, follow the instructions in [Migrating Projects from 3.1.x to 3.4](#).
- To migrate from 3.2 or 3.3 to 3.6-3.8, follow the instructions in [Migrating Projects from 3.2 or 3.3 to 3.4](#).
- To migrate from 3.5 to 3.6-3.8, see [Changes in Behavior for Migration to 3.6-3.8](#).
- No steps are required when migrating from 3.6 or 3.7 to 3.8.

### Changes in Behavior for Migration to 3.6-3.8

The following changes in behavior apply to migration from 3.5 to 3.6-3.8

- The `<resourcename>.meta.xml` file has been deprecated in General Interface 3.6-3.8, so that localized applications that use the file will fail to display localized text.

If you use `<resourcename>.meta.xml` in your application, update the application to add the locales specification in the main resource properties file. For example:


```
<data jsxid="jsxroot" jsxnamespace="propsbundle"
locales="fr,zh_TW">
<locale>
<record jsxid="Line1" jsxttext="This old village" type="jsxttext"/><record jsxid=
"Line2" jsxttext="not a single house" type="jsxttext"/><record jsxid="Line3" jsxttext=
"without persimmon trees" type="jsxttext"/><record jsxid="Dialog1" jsxttext="Dialog
Title (No Chinese or French value available)" type="jsxttext"/><record jsxid="Tools1"
jsxttext="Tools" type="jsxttext"/><record jsxid="Display English" jsxttext="Display
English"/><record jsxid="Display Chinese" jsxttext="Display Chinese"/><record jsxid=
"Display French" jsxttext="Display French"/><record jsxid="Display Default" jsxttext=
"Display Default"/><!-- demonstrates fall-through ></locale></data>
```

\* XML load API uses different protocols in 3.6-3.8 than in previous versions. For releases through GI 3.5.1, the native XML Document object is used and loads XML resources synchronously. Relative path XML resources are resolved relative to the JavaScript code or project configuration file.

For releases starting with 3.6, `jsx3.net.Request` is used to load XML resources. The major advantage is that this allows for asynchronous loading of resources. However, this also changes the relative path resolution to be relative to the launch HTML page location.

- If your application uses a relative path for the XML resource, you must update the relative path to be resolved by calling `<my_application_server>.resolveURI(<my_relative_path>)` before passing it to load method such as `Document.load()`. It is a recommended good practice to always use `resolveURI()` to specify the resource URL.
- In 3.5, a component loaded from a URI beginning with `JSXAPPS/...` has its URI resolver set to the application server. In other words, `Model.getUriResolver()` returns the server of the component when called on the component. Therefore, any path resolved against the component will resolve relative to the project directory.

Beginning with Release 3.6, a component loaded from a URI beginning with `JSXAPPS/...` has its URI resolver set to `jsx3.net.URIResolver.USER`. Therefore, any path resolved against the component will resolve relative to the user workspace.

 A URI beginning with `JSXAPPS/...` is considered to be absolute (the same as `jsxuser:///JSXAPPS/...`) and is supported only for legacy reasons.

Specifically, if you load a component from a URI beginning with `JSXAPPS/...`, paths that resolve against the component and are relative to the project directory no longer work. The following examples show methods with path parameters that are resolved against the component, such as `Model.load()`, `Cacheable.setXMLId()`, and `ToolBarButton.setImage()`.

For example, the following works in 3.5, but will not work in 3.6-3.8

```
var dlg = myApp.getBodyBlock().load("JSXAPPS/sampleApp/components/dialog.xml");

dlg.load("components/matrix.xml");
```

Instead, the recommended approach is to load all components with project-relative URIs:

```
var dlg = myApp.getBodyBlock().load("components/dialog.xml");

dlg.load("components/matrix.xml");
```

The following deprecated option uses all legacy URIs:

```
dlg = myApp.getBodyBlock().load("JSXAPPS/sampleApp/components/dialog.xml");

dlg.load("JSXAPPS/sampleApp/components/matrix.xml");
```

For `Model.load()`, which accepts an optional `URIResolver` parameter, you can explicitly specify the resolver:

```
var dlg = myApp.getBodyBlock().load("JSXAPPS/sampleApp/components/dialog.xml");

dlg.load("components/matrix.xml", true, myApp); // 3rd parameter is the resolver.
```