Protecting Namespaces for Portal Deployments

This document describes how to design TIBCO General Interface applications so that they work properly in multi-application environments, such as a portal deployment.

Version 2.0: 02-10-2006

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.

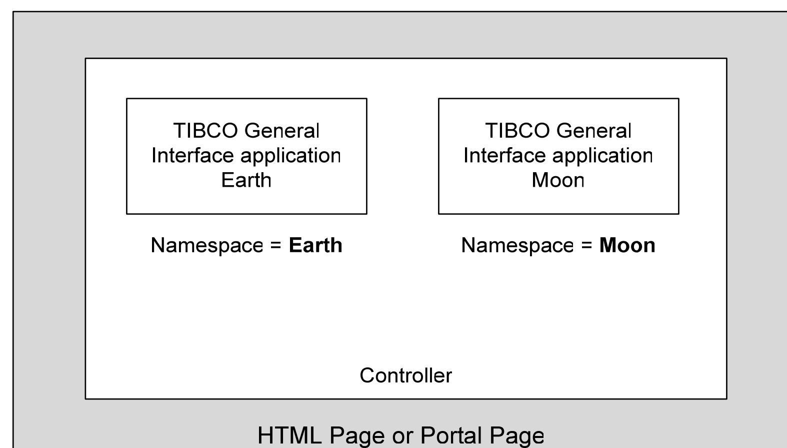
Table of Contents

How Namespaces Affect Deployed Applications 3
Setting the Application Namespace in GI Builder 4
Using the Namespace in Application Code 4
Protecting Application Code with Packages 5

How Namespaces Affect Deployed Applications

Multiple GI applications running in a single web page share a common JavaScript memory space. Consequently, one GI application can have intended and unintended effects on another GI application running in the same web page. There is no way of designing an application such that it is completely protected from other applications. This document describes how to design an application so that unintended effects between it and from other applications are greatly reduced.

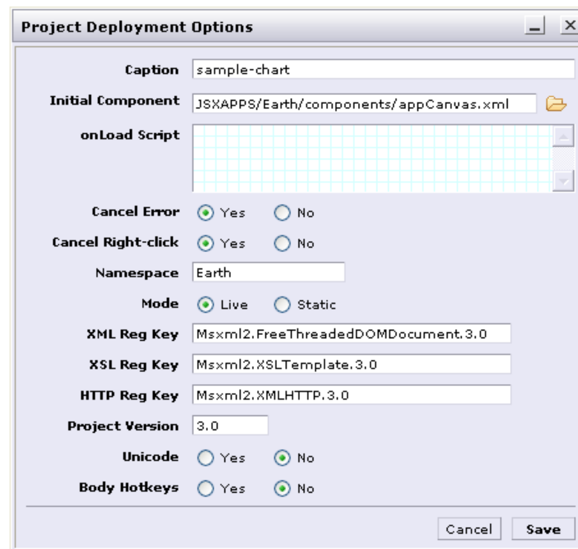
The "namespace" of a GI application is the name of the global JavaScript variable that is set to the instance of jsx3.app.Server representing the application. Custom application JavaScript code usually references this variable extensively since the server provides access to the application DOM and XML cache, among other things.



The namespace of a GI application must be unique over all the applications deployed in a single web page. If a namespace collision occurs, the code of only one of the applications will have access its jsx3.app.Server instance. The other applications will likely break. Since all deployed applications must live on disk in the same JSXAPPS folder, and each application lives in a uniquely named subfolder, setting the namespace equal to the name of the application folder is a good convention for preventing naming collisions. Namespaces should always be checked for uniqueness before creating a deployment page with multiple GI applications.

Setting the Application Namespace in GI Builder

By default, the server namespace is the same as the project name, with any illegal characters removed. The value is displayed in the Namespace field of the Project > Deployment Options dialog, and can be modified to any valid JavaScript variable name.



Using the Namespace in Application Code

The `jsx3.app.Server` instance, the name of which is specified by the application namespace, provides access to the application's DOM, and XML cache.

Access to the DOM is provided by the methods `getDOM()`, `getJSX()`, `getJSXByName()`, `getJSXById()`, and `getBodyBlock()` of the `jsx3.app.Server` class. All DOM nodes, which are instances of `jsx3.app.Model`, are indexed by both id and name. The following code retrieves the DOM node of name "textbox" in the application of namespace "Moon."

```
Moon.getJSXByName("textbox");
```

The DOM index on name only tracks one node per name. Therefore, `getJSXByName()` only returns a predictable value for names that are unique over the entire application.

The `jsx3.GO` method is a shortcut for fetching a DOM node by name. `jsx3.GO()` has access to all loaded GI applications and will look in all of them for a DOM node of the given name. `jsx3.GO` is therefore slower than `getJSXByName()` and allows one application to affect another application. It's use is only appropriate in the GI Builder scriptlet pad where brevity is more important than accuracy. Application code should not use `jsx3.GO`.

```
jsx3.GO("textbox");
```

`jsx3.GO()` accepts an optional second parameter, which specifies the namespace to look in. Calling jsx3.GO() with the second parameter does prevent it from returning a DOM node from another application. However, using `getJSXByName`() is still more efficient and is the preferred method.

```
jsx3.GO("textbox", "Moon");
```

Access to the application XML cache is provided by the `getCache()` method in the `jsx3.app.Server` class. The following code fetches a cache document from the Moon application:

```
Moon.getCache().getDocument("docId");
```

Protecting Application Code with Packages

The application namespace, described above, is not the only "namespace" that needs to be protected from other GI applications. Project JavaScript code should also be placed in a unique namespace to avoid affecting and being affected by other applications. A JavaScript namespace is known in GI as a "package."

In general applications should not declare global JavaScript functions. The following code defines a global JavaScript function:

```
function doTest() {
    return 1;
}
```

This is equivalent to declaring the function as a member of the global window object:

```
window.doTest = function() {
    return 1;
};
```

Since all GI applications deployed in the same web page share the same global window object, any code placed in it is not private to a particular application. If two different application both define a global doTest() method, one version will be overridden by the other and one application will probably break.

To avoid any naming collisions in function and variable names, all code should be placed in packages. The reverse domain naming convention is a good one for avoiding collisions. A package is essentially a nested data structure descending from the window object. The following code creates the package "com.tibco.gi" and defines the `doTest()` function in it.

```
window["com"] = new Object();
com.tibco = new Object();
com.tibco.gi = new Object();
com.tibco.gi.doTest = function() {
    return 1;
};
```

This code has a bug in it though because it will destroy any previously defined package descending from window.com, such as com.tibco.portal. With this technique for creating a package, the nested objects should only be created if they do not already exist.

The jsx3.lang.Package class simplifies the task of creating a package greatly with the `definePackage()` method. The following code also defines the `doTest()` function in the com.tibco.gi package, but without the bug in the previous code sample.

```
jsx3.lang.Package.definePackage("com.tibco.gi", function(gi) {
    gi.doTest = function() {
        return 1;
    };
});
```

Since JavaScript packages can be named using the reverse domain naming convention, they are very safe from naming collisions. Since the server namespace cannot use the reverse domain naming convention, it may be at higher

risk of a naming collision. The impact of a server namespace naming collision can be reduced by always accessing the jsx3.app.Server instance through a static method of a package. The following code defines the method com.tibco.myapp.getApp() that returns the server instance for an application with a namespace of "MyApp".

```
jsx3.lang.Package.definePackage("com.tibco.myapp", function(myapp) {
    myapp.getApp = function() {
        return MyApp;
    };
});
```

If the rest of the code in the MyApp application references the application `jsx3.app.Server` instance by `com.tibco.mypp.getApp()` rather than by MyApp, then that code will not have to change in the event of a server namespace naming collision.