

Handling Application Data

This document describes how to send, receive, and cache data. TIBCO General Interface™ provides an in-memory local data cache where developers can store XML/XSL documents needed by the application. Some documents in the local data cache are placed there automatically by the system. Developers can also set their own documents in the local data cache. Having a centralized cache provides easier object cleanup, as well as a standardized method for sharing data across multiple components. The local data cache is a critical aspect of a stateful rich client experience when used in conjunction with asynchronous data access.

Version 3.0: May 15, 2007

Scope: General Interface™ 3.4.0



<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

© 2006-2007 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

| | |
|-----------------------------------|----------|
| Overview | 3 |
| Sending Data | 4 |
| Receiving Data..... | 5 |
| Receiving XML Data | 5 |
| Handling Failures | 5 |
| Receiving Non-XML Data | 6 |
| Caching Data | 7 |
| Accessing the Correct Cache | 7 |
| Shared Global Cache..... | 7 |
| Pre-caching XML Documents | 8 |

Overview

TIBCO General Interface applications can send and receive any type of text data sent using HTTP/S. The `jsx3.net.Request` class is used for this exchange, providing a generic transport mechanism for text content, regardless of format. Data that is in XML format can be stored in the local data cache for repeated use by various objects within the application.

Sending Data

You can send data from an application using the methods: GET, POST, PUT, or DELETE.

The following describes the general pattern for sending data:

1. Create a new `jsx3.net.Request` instance.
2. Call the `open()` method and specify the interaction type (GET, POST, etc.).
3. Call the `send()` method.

For a POST interaction, pass the string content as the only parameter to the `send()` method. For GET, use another content mechanism, such as an overloaded URL.

For example:

```
//initialize
var req = new jsx3.net.Request();

//open
req.open("GET", "http://www.tibco.com?myQuery=1", true);

//create response callback and subscribe
function onResponse(objEvent) {
  var req = objEvent.target;
  alert(req.getResponseText());
};
req.subscribe(jsx3.net.Request.EVENT_ON_RESPONSE, onResponse);

//send
req.send();
```

General Interface applications perform best when they use asynchronous data access. However, there are times when you might want to synchronously access data. The above call made synchronously is as follows:

```
//initialize
var req = new jsx3.net.Request();

//open
req.open("GET", "http://www.tibco.com?myQuery=1", false);

//send
req.send();
alert(req.getResponseText());
```

To test these methods, copy and paste them into the JavaScript Test Utility (Tools > JavaScript Test Utility) in TIBCO General Interface™ Builder.

Receiving Data

The process for receiving data depends on whether the data is valid XML or another format.

Receiving XML Data

To receive XML data, call the `getResponseXML()` method on the `Request` instance. This method returns a `jsx3.xml.Document` object, which is an XML document in a format that TIBCO General Interface can use.

For example:

```
//instance the Request class
var request = new jsx3.net.Request();

//get user-specified data and open the request object
var URL = jsx3.GO('myURLComponent').getValue();
var reqSocket = request.open('GET',URL);

//send the request
reqSocket.send();

//get the XML response
var repXML = reqSocket.getResponseXML();
```

To save the response document for repeated access, use the `setDocument` function to load it into the local data cache.

For example:

```
myApp.getCache().setDocument("MyCacheDocument", repXML);
```

Also note that the `Request` class has different methods for accessing data. The example shown above is the simplest of all data calls: a synchronous HTTP GET. In fact, requesting and caching data in this manner can be done with only one API call, `openDocument`.

For example:

```
var repXML = someApp.getCache().openDocument(someURL, someCacheId);
```

Handling Failures

If the incoming XML can't be parsed by the `Request` instance, the `getResponseXML` call fails. One possible cause of failure is invalid XML, while another is unspecified *content type* specified in the HTTP header. The server must set this field in the HTTP header to `text/xml` or similar.

If the `getResponseXML` call fails but the content is valid XML, you can still parse it by calling the `getResponseText()` method and passing the XML string into a `jsx3.xml.Document` instance.

For example:

```
var objXML = new jsx3.xml.Document();
objXML.loadXML(XML_string);
if(!objXML.hasError()) alert("success:\n\n" + objXML.getXML());
```

For more details on `getResponseText`, see [Receiving Non-XML Data](#).

Receiving Non-XML Data

To receive non-XML data, call the `getResponseText()` method for the `Request` instance. This method returns the body of the response as a string value.

For example:

```
var strText = objRequest.getResponseText();  
window.alert(strText);
```

Non-XML data is useful in many situations. For example, non-XML data can be displayed in the application by setting it as the text property of a `jsx3.gui.Block` object, or it can be reformatted into CDF for use by a Matrix. Also, if the data is converted to XML, it can be stored in the local data cache to facilitate data access and management.

Caching Data

Each General Interface application maintains a separate local data cache. If two applications are embedded on a single web page, each has its own cache. This local data cache should not be confused with the browser file cache for temporary Internet files. The General Interface local data cache is an in-memory, JavaScript hash of parsed XML documents that disappears when the browser window is closed. Any XML document, including XML, CDF, and XSL files, can be stored in the local data cache.

Controls that display XML data interface with the local data cache by way of the *jsx3.xml.Cacheable* interface. This class provides methods that link a GUI object and the local data cache data it needs. The GUI object never refers to the data by reference. Only the local data cache references the XML document object, facilitating improved dereferencing and object cleanup.

The **Share Resources** property is an important property of GUI classes that extend *Cacheable* (Menu, Tree, Matrix, and so on). When the Share Resources property is set to `false` (default setting) and the GUI object is removed from the General Interface DOM, the associated XML and XSL documents are also removed from the local data cache. Set this property to `true` if two GUI objects refer to the same document or if the document should remain in the local data cache after the GUI object is removed.

Accessing the Correct Cache

To load files into the local data cache for use at runtime, call methods of the *jsx3.app.Cache* class. All cache method calls must be qualified with application namespace information. If the namespace for your application is `app1`, use the following syntax when storing and retrieving documents:

```
app1.getCache().method_name.
```

You can cache any XML, CDF, or XSL document using the `openDocument()` method. This method allows you to specify the URL of a file to load and parses the file into a *jsx3.xml.Document* instance. To cache the document, provide a second parameter to use as the cache ID,.

For example:

```
app1.getCache().openDocument("http://ibiblio.org/bosak/coriolan.xml", "someCacheId");
```

Because this method call is always synchronous, browser performance can be affected if the document is large.

To explicitly load a file into cache independently of the `openDocument()` method, use the `setDocument()` method.

To retrieve a file from cache, use the `getDocument()` method.

For example:

```
app1.getCache().getDocument("myDocument");
```

Shared Global Cache

If you need to share a cache document with other applications running in the same page, use the **shared global cache** that is provided by the General Interface system.

For example:

```
var objSharedXML = jsx3.getSharedCache().getDocument("abc");
```

When a document is requested from the local data cache and the document isn't found, the General Interface system queries the shared global cache. For example, if the local data cache doesn't have a document named `abc`, the following calls are equivalent (assuming that the application namespace is `app1`):

```
var objXML = jsx3.getSharedCache().getDocument("abc");
```

or

```
var objXML = app1.getCache().getDocument("abc");
```

Pre-caching XML Documents

In general, General Interface controls only request their XML and XSL from the local data cache when they are painted on-screen. Otherwise, accessing data is typically the domain of the developer.

In General Interface Builder™, project files can be loaded automatically when the application loads. To manually specify which project files are automatically loaded into cache, right-click the file name in the Project Files palette and select **Auto Load**. Any type of project file except a GUI component and a mapping rule can be configured to automatically load.

When you're working in TIBCO General Interface Builder, XML and XSL project files can be manually loaded or reloaded into the local data cache by doing the following:

- In the Project Files palette, right-click the file name and select **Load/Reload**.
- In the work area, right-click the associated tab and select **Save and Reload** after modifying the XML/XSL document it contains.

To view the contents of the local data cache in General Interface Builder, choose **Palettes > Local Data Cache**.