

Handling Application Data

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

Version 2.0: 02-010-2006



<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

Overview	3
Sending Data	4
Receiving Data.....	5
Receiving XML Data	5
Handling Failures	5
Receiving Non-XML Data	5
Caching Data	7
Accessing the Cache	7
Pre-caching XML Data.....	7

Overview

TIBCO General Interface applications can send and receive any type of text data sent via 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, it can be stored in the XML Cache for repeated use in 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 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(javax3.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 the developer may 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 either of these methods, paste them into the JavaScript Test Utility provided by 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();
var listView = jsx3.GO("myListComponent");

//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 XML 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`:

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

Handling Failures

If the incoming XML cannot be parsed by the `Request` instance, the `getResponseXML` call will fail. 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 `getResponseText`, and then 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, you call the `getResponseText` method for the `Request` instance. This method returns the body of the response as a string value.

For example:

Handling Application Data

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

Non-XML data is useful in many situations. For example, it can be displayed in the application by setting it as the text property on a `jsx3.gui.Block` object, or can be reformatted into CDF for use by a List or Grid. If the data is converted into XML, it can be stored in XML Cache to facilitate data access and management.

Caching Data

Each General Interface application maintains a separate XML Cache. If two applications are embedded on a single Web page, each will have its own Cache. This cache should not be confused with the browser's own file cache (i.e., temporary internet files). The General Interface XML Cache is an in-memory JavaScript hash of parsed XML documents that disappears once the browser window is closed. Any XML document, including XML, CDF, and XSL files, can be stored in the XML Cache.

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

An important property on the GUI classes that extend Cacheable (Menu, List, Tree, etc) is **Share Resources**. This flag defaults to false which means when the GUI object is removed from the General Interface DOM, its associated XML and XSL documents will also be removed from the XML Cache. If two GUI objects point to the same document or if the document should remain in the Cache after the GUI object is removed, set this property to true.

Accessing the Correct Cache

To load files into cache for use at run time, you call methods of the *jsx3.app.Cache* class. However, all cache method calls must be qualified with application namespace information. If the namespace for your application is *app1*, when storing and retrieving documents use the syntax *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 then parses the file into a *jsx3.xml.Document* instance. By providing a second parameter to use as the Cache Id, this document will also be cached.

For example:

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

This method call is always synchronous, so 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");
```

Pre-caching XML Documents

In general, General Interface controls only request their XML and XSL from the cache when they are painted on-screen. Otherwise, accessing data is typically the domain of the developer. In TIBCO General Interface Builder, project files can be loaded automatically when the application loads. To manually specify which project files are automatically loaded into cache, in the Project Files palette, right-click on the file name 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 are working in TIBCO General Interface Builder, XML and XSL project files can be manually loaded or reloaded into the XML Cache by doing the following:

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

