



TIBCO iProcess® Workspace (Browser)

Integrating TIBCO Forms 2.x with GI Applications


Version 11.10.0 | May 2025

Contents

Contents	2
Integrating TIBCO Forms 2.x with GI Applications	3
Overview	3
Forms Add-In Interface	4
Instantiating a Form	4
Providing Input Data	7
Handling Submitted Data	8
Providing Validation Methods	10
Appendix A: Complete Listing	13
TIBCO Documentation and Support Services	17
Legal and Third-Party Notices	19

Integrating TIBCO Forms 2.x with GI Applications

This document describes the integration of TIBCO Forms 2.x with TIBCO General Interface (GI) applications.

 **Note:** This document is included in the TIBCO iProcess Workspace (Browser) document set as the TIBCO Forms Add-in (the runtime part of TIBCO Forms) is installed via the TIBCO iProcess Workspace (Browser) installer.

Overview

Forms developed with TIBCO Forms are typically associated with a user task of a business process and used in the context of TIBCO iProcess Workspace (Browser) or TIBCO iProcess Workspace (Browser) Components to capture and/or display the user data. However, at times there is a need to generate customized user interfaces for user tasks that run outside the context of TIBCO iProcess Workspace (Browser), or to display data that is coming from external data channels, like TIBCO BusinessWorks web services, along with BPM data. This document describes a programmatic approach to instantiating and launching TIBCO Forms applications from a standalone General Interface application. The standalone General Interface application may be used to interact with Business Process data along with other external data channels like TIBCO BusinessWorks web services.

Restrictions

The interface described in this Tech Note applies only to the use of TIBCO Forms 2.x. The interface changes in later versions of TIBCO Forms. Any applications using TIBCO Forms in this way have to be updated to conform to the new interface.

i Note: The usage specified in this document is supported only in the context of using TIBCO iProcess suite of software, for example, for collecting data to start a business process or displaying external data referenced by the business process. This interface should not be used in any other context.

Prerequisites

It is assumed that the reader is familiar with the General Interface programming model, including the creation and manipulation of `jsx` objects (GI 3.9.0 and onwards), the container model, the event model, and making use of GI add-ins. For more information, see the GI documentation.

Forms Add-In Interface

The runtime of TIBCO Forms consists of a GI add-in that interprets the form model and instantiates the corresponding GI objects needed to render the form. The form add-in on its own does not access or submit data. The application that contains the form is responsible for retrieving data and handling submitted data. This section explains what the application needs to do in order to instantiate a form, provide initial data, and handle the submit of data from the form.

The TIBCO Forms add-in is installed as part of the TIBCO iProcess Workspace (Browser) in the following location:

```
<Tomcat Install Dir>\webApps\iprocessClient\addins\forms2
```

i Note: The **forms** addin installed at the same location is used for earlier versions of TIBCO Forms.

Instantiating a Form

The Form class is provided as `com.tibco.forms.Form`. The constructor for this class accepts a JavaScript object that contains the following properties:

Properties for the Form class constructor object

Property	Description
xml	String or <code>jsx3.xml.Document</code> . Provides the XML representing the form model. This can be provided either as a path to the form model (String) or as a parsed <code>jsx3.xml.Document</code> object.
dataStore	The instance of the <code>com.tibco.dataTIBCOItemStore</code> that extends <code>dojo.data.ItemFileWriteStore</code> (from the Dojo Toolkit). The data store is initialized with data provided by the server.
container	<code>jsx3.gui.Block</code> . This is the node in the containing application to which the Form will be added.
formLocaleString	String. This is the locale that will be used to render the form. This string is of the form <code>xx</code> or <code>xx_YY</code> , where <code>xx</code> is the language code and <code>YY</code> is the country code.
formPath	String. This is the path to the location of the form file. This is used for resolving the path to the associated resource bundles (<code>.xml</code> extension). This will be the same as the <code>xml</code> parameter if the <code>xml</code> parameter is passed in as a path to the form model.
formPathAlt	String. Alternate path to the location of the form file. Used for resolving form resources such as CSS files, image files, and external JavaScript files.
javascriptBomRoot	Path to the location of JavaScript object files created for the Business Object Model (BOM).
ignoreCache	boolean. If true, then this will force the form add-in to reload the <code>.form</code> model from the server for each form load during the browser's session. This is useful during the development cycle when the form may be undergoing many changes and you want to pick up the changes without having to re-load the entire application.
formId	String. This can be anything as long as it is unique across all the forms instantiated within the current browser session.

Property	Description
formData	The value for the property contains copy of initial data provided to the form. This data will be used during the reset of form to its original state.

Once the form is instantiated, it will be added automatically to the container provided in the Form constructor. There are still three things that need to happen after the form is instantiated:

1. Add Validation Methods to the Form (See [Providing Validation Methods](#)).
2. Add Callback Methods to the Form (See [Handling Submitted Data](#)).
3. Paint the Form. This is done using the `Form.paint()` method.

Below is a listing showing an sample function that will load the Form given a URL to a Form model:

```
function loadForm(formPath, javascriptBomRoot, sampleDataPath) {
    var pane = formViewer.getJSXByName("pane");
    _dataStore = getDataStore(sampleDataPath);
    var context = {
        formPath : formPath,
        formPathAlt : formPath,
        javascriptBomRoot : javascriptBomRoot,
        container : pane,
        locale : _locale,
        ignoreCache : false,
        formId : _formId,
        dataStore : _dataStore
    };
    var form;
    try {
        form = new com.tibco.forms.Form(context);
    } catch (e) {
        alert("error loading form: " + e );
    }
    addValidationMethods(form);
    registerCallbacks(form);
    form.paint();
    return form;
};
```

The `addValidationMethods()`, `getDataStore()`, and `registerCallbacks()` methods are explained in the following sections, as the `dataProvider` object is passed into the form constructor.

Providing Input Data

The input data is provided to the form via the Dojo data store object. You can create the data store object by providing:

- a URL to the location of the JSON data file
- or
- an array of JavaScript objects. Each object in this array will contain following properties:
 - `$param` : name of the data parameter
 - `$value` : value of the parameter
 - `type` : type of the object (primitive data type or complex data type)

Here is the listing of the `getDataStore()` method that returns the data store object using the path to the sample data file:

```
function getDataStore(sampleDataPath) {
    var logger = jsx3.util.Logger.getLogger("com.tibco.forms.app1");
    var dataStoreArgs;
    if (sampleDataPath != null && sampleDataPath.length > 0) {
        dataStoreArgs = {
            url : sampleDataPath,
            logger : logger
        };
    } else {
        var initialData = new Object();
        initialData.items = new Array();
        var dataStoreArgs = {
            data : initialData,
            logger : logger
        };
    }
    return new com.tibco.data.TIBCOItemStore(dataStoreArgs);
};
```

Here is the listing of the `getDataStore()` method that returns the data store object using the JavaScript array of items.

```
function getDataStore(items) {
    var logger = jsx3.util.Logger.getLogger("com.tibco.forms.app1");
    var dataStoreArgs;
    var initialData = new Object();
    initialData.items = items;
    var dataStoreArgs = {
        data : initialData,
        logger : logger
    };
}
return new com.tibco.data.TIBCOItemStore(dataStoreArgs);
};
```

Handling Submitted Data

The handling of form-level life-cycle events is handled by a subscribe mechanism on the Form object. The containing application registers interest in one or more of the form-level events and passes in a callback function that can retrieve parameter data from the form and complete the process of submitting the data to the server. The signature for the subscribe method:

```
subscribe(eventName, thisObject, callbackFunction);
```

Arguments for the subscribe() method

Arguments	Description
eventName	String. This is the name of the event that the listener responds to. <code>com.tibco.forms.Form</code> defines 5 event names that can be used here: SUBMIT, APPLY, CLOSE, CANCEL, RESET.
thisObject	Object. Object that is accessible as this in the callback function.
callbackFunction	A function that takes the event object and handles the particular event.

For actions where the Form should be removed from the rendered view, the `destroy()` method should be called on the Form. This will remove it from the parent and do additional clean-up. The output data can be retrieved from the Form using `Form.getDataOut()`. This object provides a `get` method that returns the value for the given

field name. Available parameter names can be accessed via `Form.getDataModel().getParamNames()`, which returns an array of parameter names.

Below is some sample code that can be used to register listeners on the Form object that will handle the form life-cycle events. For these, the handlers are only displaying the information in an alert. In a production application, the body of the submit callback would be responsible for sending the updated data back to the server.

```
function registerCallbacks(form) {
    form.subscribe(com.tibco.forms.Form.SUBMIT, form,
        function(event) {
            this.dataStoreBuff = "{ items:[ ";
            this.itemsExist = false;
            var saveCompleteCallback = function() {
                var itemCallback = function(item, request) {
                    var itemStr = _dataStore.serializeItem(item);
                    if (itemStr == "") {
                        return;
                    }
                    if (this.itemsExist) {
                        this.dataStoreBuff = this.dataStoreBuff + "," +
                            itemStr;
                    } else {
                        this.dataStoreBuff = this.dataStoreBuff +
                            itemStr;
                        this.itemsExist = true;
                    }
                    alert("submitting param: " +
                        _dataStore.getValue(item, "$param") + "=" +
                        itemStr);
                };
                _dataStore.fetch( {
                    query : {
                        $param : "*"
                    },
                    onItem : itemCallback,
                    scope : this
                });
            };
            _dataStore.save( {
                "onComplete" : saveCompleteCallback,
                "scope" : this
            });
            this.dataStoreBuff = this.dataStoreBuff + " ]} ";
            alert("Data store output submitted : " + this.dataStoreBuff);
        });
    form.subscribe(com.tibco.forms.Form.CLOSE, form,
        function(event) {
```

```

        alert("Close Action invoked");
    });
    form.subscribe(com.tibco.forms.Form.CANCEL, form,
        function(event) {
            alert("Cancel Action invoked");
        });
    form.subscribe(com.tibco.forms.Form.APPLY, form,
        function(event) {
            alert("Apply Action invoked");
        });
    });
};

```

Providing Validation Methods

Before the form can be put into service, there are three validation methods that the runtime expects to be added to the form. The following code can be used to add the required methods to the form after instantiation:

```

adapter.prototype.addValidationMethods = function(form) {
    form.maxLength = function(value, len) {
        return value.length <= len;
    };
    form.isNumber = function(value, length) {
        if (!value || value == null) {
            return true;
        }
        if (typeof (value) == 'number') {
            value = value + "";
        }
        if (value.length == 0) {
            return true;
        }
        if (value.indexOf(".") != -1) {
            var index = value.indexOf(".");
            var fraction = value.substring(index + 1);
            value = value.substring(0, index - 1);
            if (fraction.valueOf() != 0) {
                return false;
            }
        }
        if (isNaN(value)) {
            return false;
        }
        if (/^~?\d|E|e+$/i.test(value)) {
            if (value.indexOf('-') == -1 && value.length > length) {

```

```

        return false;
    } else {
        return true;
    }
} else {
    return false;
};
};
form._numberFormat = function(value, len, digits) {
    if (typeof (value) == 'number') {
        value = value + "";
    }
    if (!value || value == null || value.length == 0) {
        return true;
    }
    if (isNaN(value)) {
        return false;
    }
    var whole;
    var fraction;
    var index = value.indexOf('.');
    if (index == -1) {
        whole = value;
        fraction = '';
    } else {
        whole = value.substring(0, index);
        fraction = value.substring(index + 1);
        if (digits == 0 && fraction.valueOf() == 0) {
            fraction = '';
        }
    }
    var wholeLength = len;
    if (digits != -1)
        var wholeLength = len - digits;
    if (whole.indexOf('-') == 0) {
        wholeLength++;
    }
    return (this.maxLength(whole, wholeLength)
    && (digits != -1 ? this.maxLength(fraction, digits) : true)
    && this.isNumber(whole) && this.isNumber(fraction));
};
form.numberFormat = function(value, len, digits) {
    if (value instanceof Array) {
        var result = true;
        for ( var i = 0; i < value.length; i++) {
            result = this._numberFormat(value[i], len, digits);
            if (result)
                continue;
        }
    }
};

```

```
        else
            return result;
        }
        return result;
    } else {
        return this._numberFormat(value, len, digits);
    }
};
};
```

Appendix A: Complete Listing

A complete listing of the `logic.js` for a sample GI application is provided in this appendix.

```
jsx3.require("com.tibco.forms.Form");
_locale = "en_US";
_formId = 0;
_form = null;
_dataStore = null;
/*
 * This API is invoked from the containing application.
 * @param formPath: Path to location of form
 * @param javascriptBomRoot: Path to the location of the JavaScript
objects
 * @param sampleDataPath: Path to the location of sample data file
 */
function loadTIBCOForm(formPath, javascriptBomRoot, sampleDataPath) {
    if (_form != null) {
        _form.destroy();
    }
    _form = loadForm(formPath, javascriptBomRoot, sampleDataPath);
};
function loadForm(formPath, javascriptBomRoot, sampleDataPath) {
    var pane = formViewer.getJSXByName("pane");
    _dataStore = getDataStore(sampleDataPath);
    var context = {
        formPath : formPath,
        formPathAlt : formPath,
        javascriptBomRoot : javascriptBomRoot,
        container : pane,
        locale : _locale,
        ignoreCache : false,
        formId : _formId,
        dataStore : _dataStore
    };
    var form;
    try {
        form = new com.tibco.forms.Form(context);
    } catch (e) {
        alert("error loading form: " + e );
    }
    addValidationMethods(form);
    registerCallbacks(form);
    form.paint();
}
```

```

        return form;
    };
    function getDataStore(sampleDataPath) {
        var logger = jsx3.util.Logger.getLogger("com.tibco.forms.app1");
        var dataStoreArgs;
        if (sampleDataPath != null && sampleDataPath.length > 0) {
            dataStoreArgs = {
                url : sampleDataPath,
                logger : logger
            };
        } else {
            var initialData = new Object();
            initialData.items = new Array();
            var dataStoreArgs = {
                data : initialData,
                logger : logger
            };
        }
        return new com.tibco.data.TIBCOItemStore(dataStoreArgs);
    };
    function registerCallbacks(form) {
        form.subscribe(com.tibco.forms.Form.SUBMIT, form, function(event) {
            this.dataStoreBuff = "{ items:[ ";
            this.itemsExist = false;
            var saveCompleteCallback = function() {
                var itemCallback = function(item, request) {
                    var itemStr = _dataStore.serializeItem(item);
                    if (itemStr == "") {
                        return;
                    }
                    if (this.itemsExist) {
                        this.dataStoreBuff = this.dataStoreBuff + "," +
                            itemStr;
                    } else {
                        this.dataStoreBuff = this.dataStoreBuff +
                            itemStr;
                        this.itemsExist = true;
                    }
                }
                alert("submitting param: " +
                    _dataStore.getValue(item, "$param") + "=" +
                    itemStr);
            };
            _dataStore.fetch( {
                query : {
                    $param : "*"
                },
                onItem : itemCallback,
                scope : this
            });
        });
    }

```

```

        });
    };
    _dataStore.save( {
        "onComplete" : saveCompleteCallback,
        "scope" : this
    });
    this.dataStoreBuff = this.dataStoreBuff + " ]} ";
    alert("Data store output submitted : " +
        this.dataStoreBuff);
});
form.subscribe(com.tibco.forms.Form.CLOSE, form,
    function(event) {
        alert("Close Action invoked");
    });
form.subscribe(com.tibco.forms.Form.CANCEL, form,
    function(event) {
        alert("Cancel Action invoked");
    });
form.subscribe(com.tibco.forms.Form.APPLY, form,
    function(event) {
        alert("Apply Action invoked");
    });
};
function addValidationMethods(form) {
    form.maxLength = function(value, len) {
        return value.length <= len;
    };
    form.isNumber = function(value, length) {
        if (!value || value == null) { return true; }
        if (typeof (value) == 'number') {
            value = value + "";
        }
        if (value.length == 0){
            return true;
        }
        if (value.indexOf(".") != -1) {
            return false;
        }
        if (isNaN(value)) {
            return false;
        }
        if (/^-?\d|E|e+$/ .test(value)) {
            if (value.indexOf('-') == -1 && value.length > length) {
                return false;
            } else {
                return true;
            }
        } else {
            return true;
        }
    } else {

```

```

        return false;
    };
};
form._numberFormat = function(value, len, digits) {
    if (typeof (value) == 'number') {
        value = value + "";
    }
    if (!value || value == null || value.length == 0) {
        return true;
    }
    if (isNaN(value)) {
        return false;
    }
    var whole;
    var fraction;
    var index = value.indexOf('.');
    if (index == -1) {
        whole = value;
        fraction = '';
    } else {
        whole = value.substring(0, index);
        fraction = value.substring(index + 1);
    }
    var wholeLength = len - digits;
    if (whole.indexOf('-') == 0) {
        wholeLength++;
    }
    return (this.maxLength(whole, wholeLength) &&
        this.maxLength(fraction, digits) &&
        this.isNumber(whole) && this.isNumber(fraction));
};
form.numberFormat = function(value, len, digits) {
    if (value instanceof Array) {
        var result = true;
        for ( var i = 0; i < value.length; i++) {
            result = this._numberFormat(value[i], len, digits);
            if (result)
                continue;
            else
                return result;
        }
        return result;
    } else {
        return this._numberFormat(value, len, digits);
    }
};
};
};

```


TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO iProcess® Workspace (Browser) is available on the [TIBCO iProcess® Workspace \(Browser\) Product Documentation](#) page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature

requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document 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 Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, and iProcess are either registered trademarks or trademarks of Cloud Software Group, 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. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG 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” file for the availability of a specific version of Cloud SG software 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. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

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 "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2005-2025. Cloud Software Group, Inc. All Rights Reserved.