TIBCO® BPM Enterprise

Developer's Guide

Version 5.6.0 | November 2024

Copyright © 2015-2024. Cloud Software Group, Inc. All Rights Reserved.

Contents

Contents **2**

Client Application Development **4**

Access API Explorer **5**

Working With Forms **6**

Overview 6

Using TIBCO Forms 6

Displaying a Work Item Form 7

Open a Work Item Associated with a Pageflow 11

Integrating TIBCO Forms with Custom Client Applications 19

- Injecting the Forms Runtime Adapter in the Browser 19
- Typical Flow of Events 20
- iFrame Capability 24
- iframe Integration 24
- Forms Runtime Adapter 25

Working With Business Services **40**

Starting a Business Service 40

- List Business Services 44
- Start Business Service 45
- Update Business Service 46
- Cancel Business Service 48

Working With Case Data **49**

Query and Fetch Case Data Types 49

- List Case types 53
- List Cases for Case Type 56

List Case Actions 57
Read Cases 59
Conventional Database View of a Case Type 59

Working with Process Manager 61
Creating an Instance from a Specified Process Template 61
Sending an Event to a Process 64

Working with Work Item 66
Work Item States 66
Work Item State Transitions 67

Applications 73
How to Access Application Development 73
Application Lifecycle 74

REST API 76
Authentication 77
SAML Web Profile Authentication 78
OpenID Connect Authentication 79

TIBCO Documentation and Support Services 82

Legal and Third-Party Notices 84

Client Application Development

TIBCO BPM Enterprise provides Application Development to create, develop, and test custom client applications hosted in TIBCO BPM Enterprise.

Upload application files to Application Development, and then edit, test, and verify changes. For example, keep the service logic of the worklist, but completely change the appearance of the layout.

You can customize applications by adding a company logo, and incorporating the company's color scheme. See the "Customizing your Application" topic in the *TIBCO BPM Enterprise Administration Guide*. The custom application is available to users immediately after it is published.

You can delete the applications and download the application content in a .zip file.

Access API Explorer

Perform the following steps to access the API Explorer in TIBCO BPM Enterprise:

Procedure

1. Enter the following URL in your browser:

   ```
   protocol://host:port/apps/login
   ```

   - Here, *protocol* is the communications protocol that is used (`http` or `https`).
   - *host* is the DNS name or IP address of the server hosting the TIBCO BPM Enterprise runtime.
   - *port* is the port that is used. (**Default:** 80)
2. Log in with a valid TIBCO BPM Enterprise username and password.
3. To access API Explorer, click **App Switcher** .
4. In the **App Switcher** menu, click **API Explorer**.

Working With Forms

TIBCO forms will be displayed in Client Applications for

- work items,
- work items with pageflow (using page activities),
- and business services (using page activities).

Overview

A client application may need to display a form to a user when that user opens a work item that starts a pageflow, or starts a business service.

The client application can render the form using either:

- a TIBCO form, which provides a web-based user interface to the work item data. See Using TIBCO Forms.
- a custom form developed as part of the client application.

Using TIBCO Forms

TIBCO forms provide web-based user interfaces for business processes. At design-time, process analysts and solution designers can use TIBCO Business Studio - BPM Edition to design forms and associate them with user tasks in business processes or pageflow processes.

See the TIBCO Business Studio - BPM Edition documentation for more information.

TIBCO Business Studio - BPM Edition generates run-time implementations of these forms based on the use of channel types and presentation channels:

- A *channel type* defines a method employed to deliver and display a form to a user. The specification of a channel type defines:

    - a delivery mechanism (for example, web client or email).
    - a rendering technology (Google Web Toolkit (GWT).
- A *presentation channel* is a container for a selection of available channel types; it defines how a form can be delivered and presented to users.

TIBCO Business Studio - BPM Edition automatically generates an implementation of each form defined in a project for each channel type defined in the project's presentation channel(s).

> **Note:** If no form has been defined for a user activity, default form implementations are automatically generated for the default presentation channel.

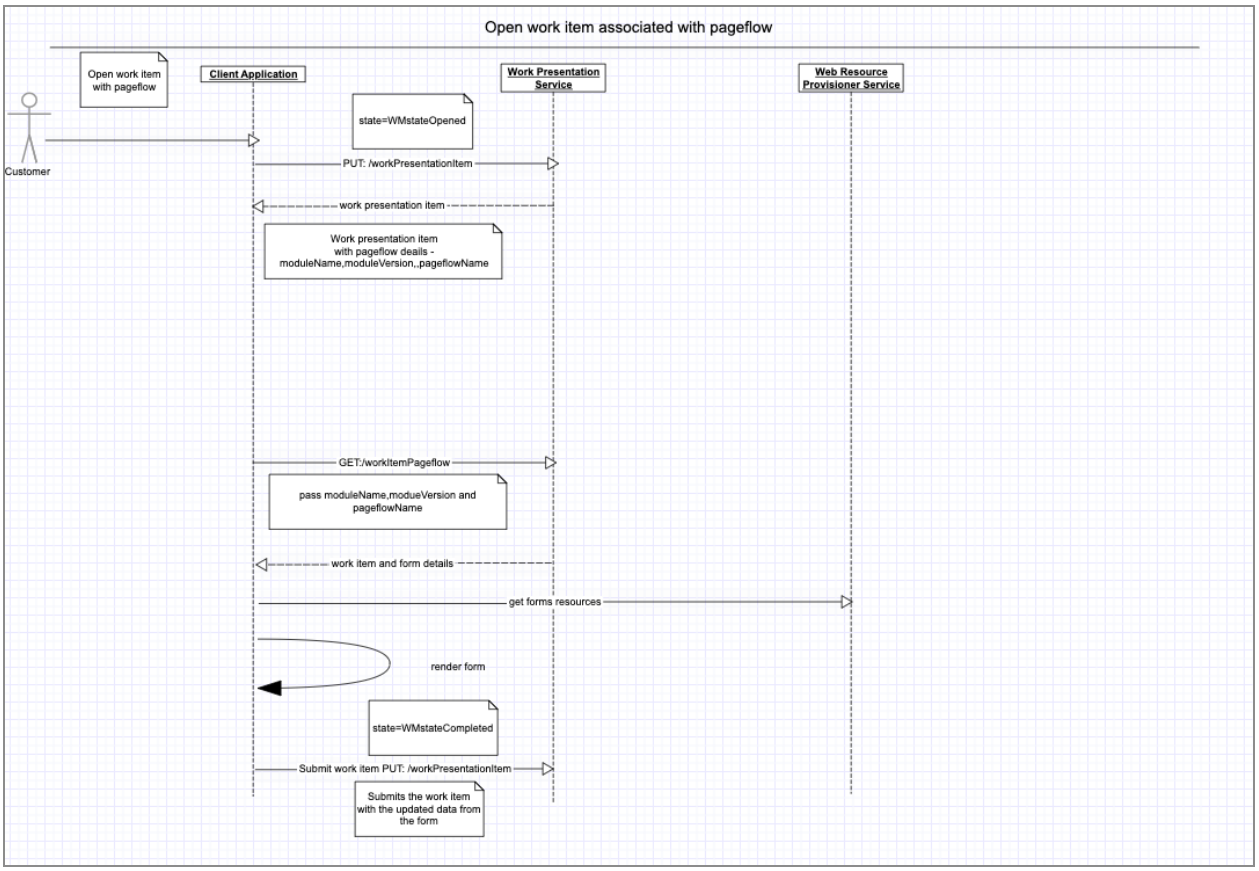
When a project is deployed to the BPM runtime, the generated form artifacts are deployed as well. (**bpmresources** is the name of the BPM runtime component that provides access to the form resources of deployed BPM applications.)

A client application can use the `WorkPresentationService` to access the form artifacts that have been deployed to the BPM runtime, and so display an appropriate TIBCO form for a work item or pageflow/business service page.

Displaying a Work Item Form

This topic provides an example that shows the sequence of calls a client application should make to display a work item form.

> **Note:** The following step-by-step description corresponds to the process shown in the diagram. The descriptions are from the perspective of the REST API. The process of performing operations using the REST API is explained with the help of an example.



When a user opens a work item from the user interface:

Procedure

1. A work item is opened by invoking a PUT call on work presentation REST service.
    - Pass the work id, version along with the payload, and the state `WMstateOpen` to open the work item.
    - `/Workpresentationitem` updates and opens the specified work item.
    - Payload for the PUT calls is similar to the one shown below:

```
{
 "id": "string",
  "version": "string",
  "header": {
    "name": "string",
    "description": "string",
    "itemContext": {
```

```
      "activityId": "string",
      "activityName": "string",
      "appInstance": "string",
      "appName": "string",
      "appVersion": 0,
      "appId": "string",
      "appInstanceDescription": "string",
      "processName": "string",
      "caseRef": "string"
    },
    "workType": {
      "uid": "string",
      "version": "string"
    }
  },
  "state": {
    "stringKey": "WMstateOpen",
    "description": "string"
  },
  "body": {
    "inputs": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
        "structured": [
          {

          }
        ]
      }
    ],
    "outputs": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
```

```
        "structured": [
          {

          }
        ]
      }
    ],
    "inouts": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
        "structured": [
          {

          }
        ]
      }
    ]
  },
  "presentation": {
    "channelId": "string",
    "channelType": "string",
    "formIdentifier": "string",
    "pageflow": {
      "moduleName": "string",
      "processName": "string",
      "version": "string"
    },
    "worktypeId": "string",
    "worktypeVersion": "string"
  }
}
```

Response of the above calls returns the updated payload and also the form details associated with the work item task

2. The client uses the form details and renders the form using the forms runtime loaded in the client. The user interacts with the form by entering details and when the form is submitted or closed, the work form returns the updated data to the client.
3. The client invokes work presentation PUT call again passing the payload received

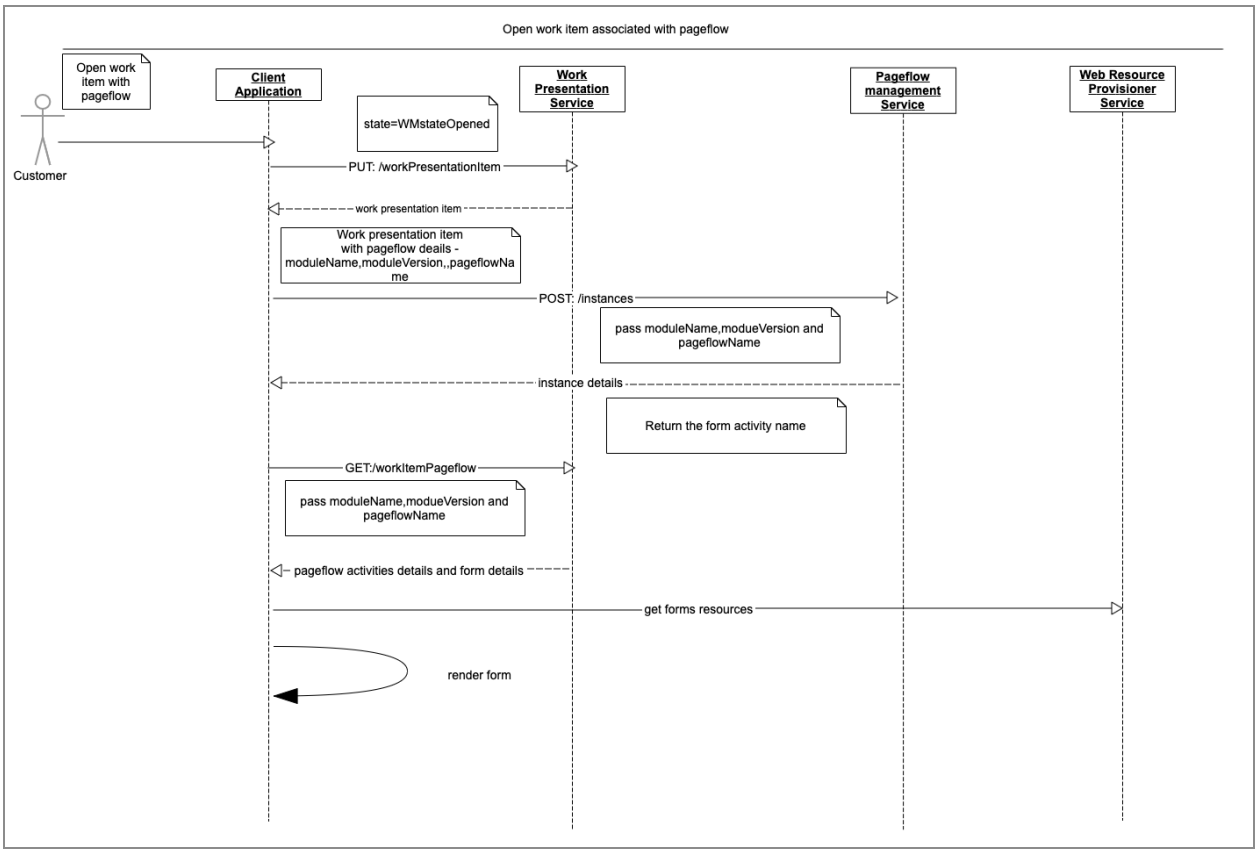
from the form and passing appropriate work item state depending on if the form is closed or submitted. When closing the work item `WMstateClose` is passed and when submitting the form `WMstateCompleted` is passed.

- `WMstateClosed`
- `WMstateCompleted`

Open a Work Item Associated with a Pageflow

This topic includes an example that shows the sequence of calls that a client application should make to open a work item associated with a pageflow.

Note: The following description corresponds to the process shown in the diagram. The descriptions are from the perspective of the REST API. The process of performing operations using the REST API is explained with the help of an example.



Perform the following steps to open a work item by using REST API:

Procedure

1. Open a work item by invoking a REST API call on the work presentation service.
    - Invoke a PUT method on REST API `/Workpresentationitem` by passing the work id, version, payload, and the `WMstateOpen` state

```
{
 "id": "string",
  "version": "string",
  "header": {
    "name": "string",
    "description": "string",
    "itemContext": {
      "activityId": "string",
      "activityName": "string",
      "appInstance": "string",
      "appName": "string",
      "appVersion": 0,
      "appId": "string",
      "appInstanceDescription": "string",
      "processName": "string",
      "caseRef": "string"
    },
    "workType": {
      "uid": "string",
      "version": "string"
    }
  },
  "state": {
    "stringKey": "WMstateOpen",
    "description": "string"
  },
  "body": {
    "inputs": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
        "structured": [
          {
```

```
          }
        ]
      }
    ],
    "outputs": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
        "structured": [
          {

          }
        ]
      }
    ],
    "inouts": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
        "structured": [
          {

          }
        ]
      }
    ]
  },
  "presentation": {
    "channelId": "string",
    "channelType": "string",
    "formIdentifier": "string",
    "pageflow": {
      "moduleName": "string",
      "processName": "string",
```

```
      "version": "string"
    },
    "worktypeId": "string",
    "worktypeVersion": "string"
  }
}
```

The response of the above calls returns the updated payload and the form details associated with the work item task. If the work item is associated with a pageflow, pageflow details are returned as part of the open work item response.

2. Based on the pageflow details returned as part of the work item response, the client application invokes Pageflow POST REST API instance call to start the pageflow. For details, click **API Explorer** > **Process Management Service**.
   - `/Instances` creates an instance in the specified pageflow process.
   - Work item data returned from the work presentation call is passed in as data to pageflow call. The module below is passed as part of the payload to the REST call that returned in the open work item call.

     ```
     {
       "moduleName": "string",
       "processName": "string",
       "moduleVersion": "string",
       "data": "string"
     }
     ```

   - Response of the Pageflow call returns the pageflow details and the activity information paused at the Activity pageflow. The response returns the state of the pageflow.

     ```
     {
       "instanceId": "string",
       "processId": "string",
       "moduleName": "string",
       "moduleVersion": "string",
       "processName": "string",
       "processLabel": "string",
       "state": {
         "state": "STARTING",
         "failedMessage": {
     ```

```
      "msgName": "START",
      "exceptionMsg": "string",
      "exceptionStack": "string",
      "msgData": "string",
      "activityData": "string"
    }
  },
  "activityInfo": {
    "activityId": "string",
    "activityName": "string",
    "activityProcessPackageId": "string",
    "activityProcessModuleName": "string",
    "activityProcessModuleVersion": "string",
    "activityPageflowName": "string",
    "activityData": "string"
  }
}
```

3. To retrieve the form details of the activity, the client application invokes the work presentation GET work item pageflow call. For details, click **API Explorer** > **Presentation Management Service**.
   - `/WorkItemPageFlow` is the Pageflow associated with the work item.
   - The response of this REST call returns the form details as a form identifier.

```
{
  "moduleName": "string",
  "processName": "string",
  "moduleVersion": "string",
  "pageflowActivities": [
    {
      "activityName": "string",
      "activityId": "string",
      "formIdentifier": "string"
    }
  ]
}
```

   - The client application uses the form identifier and renders the form using forms runtime.
4. The user interacts with the forms and enters the data in the form and when the form is closed or completed, Pageflow PUT call is called to progress the pageflow. The

data returned from the form is passed in as a payload to Pageflow call.

- `/Instances` updates a specified Process Instance.

```
{
"instanceId": "string",
  "processId": "string",
  "moduleName": "string",
  "moduleVersion": "string",
  "processName": "string",
  "processLabel": "string",
  "state": {
    "state": "STARTING",
    "failedMessage": {
      "msgName": "START",
      "exceptionMsg": "string",
      "exceptionStack": "string",
      "msgData": "string",
      "activityData": "string"
    }
  },
  "activityInfo": {
    "activityId": "string",
    "activityName": "string",
    "activityProcessPackageId": "string",
    "activityProcessModuleName": "string",
    "activityProcessModuleVersion": "string",
    "activityPageflowName": "string",
    "activityData": "string"
  }
}
```

5. After the pageflow execution is complete, this information of pageflow state can be found in the state attribute of the pageflow response. Work presentation PUT call is invoked to complete the work item which passes the payload that is received from the form. It also passes the appropriate work item state depending on if the form is closed or submitted. When closing the work item, `WMstateClose` is passed and when submitting the form, `WMstateCompleted` is passed.

   - `/Workpresentationitem` updates the specified work item

```
{
 "id": "string",
  "version": "string",
```

```
  "header": {
    "name": "string",
    "description": "string",
    "itemContext": {
      "activityId": "string",
      "activityName": "string",
      "appInstance": "string",
      "appName": "string",
      "appVersion": 0,
      "appId": "string",
      "appInstanceDescription": "string",
      "processName": "string",
      "caseRef": "string"
    },
    "workType": {
      "uid": "string",
      "version": "string"
    }
  },
  "state": {
    "stringKey": "WMstateCompleted",
    "description": "string"
  },
  "body": {
    "inputs": [
      {
        "name": "string",
        "simple": [
          {

          }
        ],
        "structured": [
          {

          }
        ]
      }
    ],
    "outputs": [
      {
        "name": "string",
        "simple": [
```

```
            {

            }
          ],
          "structured": [
            {

            }
          ]
        }
      ],
      "inouts": [
        {
          "name": "string",
          "simple": [
            {

            }
          ],
          "structured": [
            {

            }
          ]
        }
      ]
    },
    "presentation": {
      "channelId": "string",
      "channelType": "string",
      "formIdentifier": "string",
      "pageflow": {
        "moduleName": "string",
        "processName": "string",
        "version": "string"
      },
      "worktypeId": "string",
      "worktypeVersion": "string"
    }
}
```

Integrating TIBCO Forms with Custom Client Applications

TIBCO forms are a part of projects that are deployed to the runtime.

The key top-level components that are involved in making TIBCO Forms accessible from custom client applications are as follows:

- **BPM REST Services API** - BPM exposes its functionality through comprehensive work management and process management APIs. The main services that are used in the form applications refer to the **API Explorer** link. A client application can access the BPM Rest API.
- **Client Application** - The client application is typically a web application that bundles together the non-Form UI resources that are used by the application, and a controller servlet that acts as the conduit between the browser and the BPM Rest API.

Injecting the Forms Runtime Adapter in the Browser

The Forms Runtime Adapter is hosted by the BPM runtime. To use this, you first need to load the Forms Runtime Adapter file on the page. This is typically done via a `<script>` tag in the `<head>` section of the HTML document.

```
<script src="http://<host-name>:<port>/apps/bpm-
forms/formsclient/formsclient.nocache.js"/>
```

Once the JavaScript API loading is complete, it notifies the client application by invoking a function called onTIBCOFormRunnerLoad. The client application can define this function on the page and receive notification of the availability of com.tibco.forms.client.FormRunner.

For example:

```
function onTIBCOFormRunnerLoad() {
    // Forms Runtime Adapter is now available for use on the page.
    // com.tibco.forms.client.FormRunner.loadForm() or the custom
    // element named 'tibco-form' can be accessed from now on
```

```
    // to load the form.
}
```

Typical Flow of Events

This topic describes the typical flow of events when a user opens a work item from the worklist displayed in the browser of the client application.

The details of the events are as follows:

1. To open a work item, the client application invokes **Work Presentation REST API**. For details, please refer to the **API Explorer** link.
2. In response to **Work Presentation's Work Item API**, the `formUrl` and `formData` are returned to the client application. The client application decides the `locale` for rendering the form and provides the identifier of the DOM node under which the form will be rendered. For more information, refer to the **API Explorer** link.

   The client application invokes the FormRunner.loadForm() method, which accepts the above values along with two arguments: `onSuccess` and `onError`. See com.tibco.forms.client.FormRunner for details of the `FormRunner.loadForm()` method.

   You can access TIBCO Forms custom element using the tag 'tibco-form'.

```
<tibco-form
        formurl=<url_to_the_form_psm_model>
        initdata=<form_initial_data>
        bomjspath=<bom_js_path>
        locale=<locale_to_be_used>
        onload="alert('Form loaded')"
        onsubmit="event.detail.form.getSerializedParameters(formData
        => alert(formData))"
        onclose="event.detail.form.getSerializedParameters(formData
        => alert(formData))"
        oncancel="event.detail.form.getSerializedParameters(formData
        => alert(formData))">
```

```
</tibco-form>
```

Standard DOM APIs can be used to create this element. For example, document.createElement ("tibco-form"). Currently, direct usage of this tag in the markup is not allowed. The 'tibco-form' tag supports the following properties and events:

Property	Description
formurl	Pass the Form URL obtained from the work item.
initdata	Pass the initial form data obtained from the work item.
initdataurl	If the initial data is available on the server in a file, use its URL.
bomjspath	Pass the BOM JavaScript root path obtained from the work item.
locale	The locale to be used in the form. Pass the application locale, if any.

Event	Description
load	Event triggered when the form loading completes. The data returned to the event handler has the reference to the form via data.detail.form.
loadError	Event triggered when the form loading fails. The data returned to the event handler contains an error in the attribute data.detail.message.
autofocus	The autofocus value can be *true* or *false*. If the autofocus value is *true* when the form is loaded, the focus will be on the form elements. By default, the autofocus value is *false*.
useiframe	The useiframe value can be *true* or *false*.

Event	Description
	If the useiframe value is *true* when the form is loaded, it will lead to the usage of an inline frame to load the form. By default, the default value is *false*. For more details, see iFrame Capability.
cancel	Event triggered when the form is canceled. The data returned to the event handler has the reference to the form via data.detail.form.
close	Event triggered when the form is closed. The data returned to the event handler has the reference to the form via data.detail.form.
submit	Event triggered when the form is submitted. The data returned to the event handler has the reference to the form via data.detail.form.

Alternatively, the following code snippet shows how the form is loaded using the `FormRunner.loadForm()`method.

```
var formURL; // Form URL obtained from the work item.
var formData; // The initial form data obtained from the work item.
var bomJSPath; // BOM JavaScript root path obtained from the work
item.
var locale = "en_US"; // locale to use
var parentId; // Identifier of a node to which the form is added.
var submitHandler = function(actionName, form) {
    var formData = form.getSerializedParameters();
    // submit the work item
           form.destroy();
};
var closeHandler = function(actionName, form) {
    // close the work item
           form.destroy();
};
var cancelHandler = function(actionName, form) {
```

```
    // cancel the work item
           form.destroy();
};
var onSuccess = function(form) {
    form.setActionHandler(com.tibco.forms.client.Form
        .ACTION_SUBMIT, submitHandler);
    form.setActionHandler(com.tibco.forms.client.Form
        .ACTION_CLOSE, closeHandler);
    form.setActionHandler(com.tibco.forms.client.Form
        .ACTION_CANCEL, cancelHandler);
};
var onError = function(e) {
    alert("An error occurred while loading the form: "+ e);
};
com.tibco.forms.client.FormRunner.loadForm(formURL, formData,
bomJSPath, locale, parentId, onSuccess, onError, JSONP);
```

Note: Avoid using the `com.tibco.forms.client.FormRunner.loadForm()` method on the page `onLoad` event, as the `com.tibco.forms.client.FormRunner` class might not be fully loaded and the API methods being used might not be available. To avoid these errors, the client application can define the `onTIBCOFormRunnerLoad` function on the page and can be notified about the availability of `com.tibco.forms.client.FormRunner`. For more details, see Injecting the Forms Runtime Adapter in the Browser.

3. The form is displayed in the browser. The `FormRunner.loadForm()` method is asynchronous, so any post-processing that is done on the loaded form object happens within the `onSuccess` callback handler. In the above code snippet, three action handlers are set that handle the submit, close, and cancel operations that are provided on most forms.
4. When the form is submitted, the `submitHandler` handles the submit action. In response to form submission, `form.getSerializedParameters()` method retrieves the `formData`. See com.tibco.forms.client.Form for details of the `form.getSerializedParameters()` method. This method returns a JSON (JavaScript Object Notation) serialization of the data within the form.
5. After successful submission of the form, the client application invokes

completeWorkItem to pass the form data back to the WorkPresentationService. This function is used to update the work item.

iFrame Capability

A useful feature of the tibco-form element is its ability to load the form in an inline frame, or, iFrame. This is especially useful in cases where the application has many different CSS files that can conflict with the styles defined in the Form related CSS files.

To enable this feature, set the attribute as *useiframe="true"*. To load a form in an iframe, refer to the following sample:

```
<tibco-form
json2form="true"
json2formdata='{"name":"user_name", "company": "company_name",
"country": "country_name"}'
useiframe="true">
</tibco-form>
```

For best results, use this approach for iFrame support.

iframe Integration

A form can be loaded in an inline frame (`iframe`) in a custom application. The `loadForm` APIs in the `com.tibco.forms.client.FormRunner` class support the parent node ID of an <iframe> element.

To make this integration easy, the file `IFrameSource.html` is provided in the forms client (which is bundled inside the Forms Runtime Adapter). An `iframe` in the custom application can use the following source URL:

```
<iframe class="tf-form-not-loaded" id="formContainer"
src="/bpmresources/formsclient/IFrameSource.html">
</iframe>
```

Using `IFrameSource.html` is optional. Alternatively, you can provide a custom `iframe` source page, if desired.

The forms runtime loaded in a parent window talks to the forms runtime loaded in the `iframe`. For this communication to happen, the `formsclient.nocache.js` file must be loaded. The `IFrameSource.html` page loads `formsclient.nocach.js` by default. However, if you provide a custom `iframe` source page, ensure that it loads `formsclient.nocache.js` (using a `<script>` tag in the HTML. See Injecting the Forms Runtime Adapter in the Browser).

When the form is loaded in the `iframe`, the Forms Runtime Adapter removes the class selector `tf-form-not-loaded` from the `<iframe>` element and adds another class, `tf-form-loaded`. Similarly, when the form is destroyed by the Cancel, Close, or Submit actions, the Forms Runtime Adapter adds the class `tf-form-not-loaded` back to the `<iframe>` element. This can be used to control the visibility of the `iframe` within a custom application.

Note that changing the CSS class name based on whether a form is currently loaded or not is taken care of by the `formsclient` loaded on the parent window (therefore, your `iframe` element can initially have the `class tf-form-not-loaded`, which is automatically removed when the form is loaded).

The area of the `iframe` where the form is loaded is based on the following conditions:

- if the same `parentNodeId` is available within the `iframe`, the form is loaded within the `iframe`.
- if the `iframe` has an element with the ID `tfFormContainer`, the form is loaded in that element.
- if both of the above conditions are missing, the form is loaded directly under the body element within the `iframe`.

Forms Runtime Adapter

The Forms Runtime Adapter provides access to methods for instantiating, accessing, and measuring the performance of forms.

The Forms Runtime Adapter defines the following three classes:

- com.tibco.forms.client.FormRunner
- com.tibco.forms.client.Form
- com.tibco.forms.client.LoadStat

com.tibco.forms.client.FormRunner

The com.tibco.forms.client.FormRunner class provides static utility methods for instantiating the Form class in the custom client application. It also provides access to the Forms logger, which can be used even when no forms are loaded.

For more details on the `logger()` method, see the table at the bottom of this page.

The following methods are supported:

- `loadForm()`
- `loadFormWithRemoteData()`

The `loadForm()` and `loadFormWithRemoteData()` methods have the same set of input parameters but they differ in the way the initial data are passed. Both the methods are void.

- In `loadForm()`, the initial data are passed directly as a JSON (JavaScript Object Notation) string.
- In `loadFormWithRemoteData()`, a URL of the initial JSON data is passed using the `initDataURL` parameter. The FormRunner retrieves the data from the specified URL.

The details of these methods are as follows:

com.tibco.forms.client.FormRunner Class

Method	Description
`loadForm( String url, String initialData, String bomJSPath, String locale, String parentNodeId, Function onSuccess, Function onError, Boolean JSONP)`	Loads the form at the specified URL. The parameter details are as follows: • `url` - Specifies the URL to the form JSON representation, e.g. "`http://<hostname:port> /bpmresources/com.example.myproject_ 1.0.0.20201105175821853/wp/openspaceGWTPull_ DefaultChannel/openspaceGWTPull_ DefaultChannel/FindAddress/GetAddress/Getuserdetails/ Getuserdetails.gwt.json`" • `initialData` - Specifies the JSON representation of the initial data that are provided to the form. • `bomJSPath` - Specifies the root folder path used for loading

Method	Description
	the BOM JavaScript files used by the form, e.g. "`http://<hostname:port>/bpmresources`" • `locale` - Specifies the locale to be used in the form runtime with format <lc>[_<CC>] where [] denotes optionality, e.g. "`en_US`". The locale needs to be represented such that <lc> is a valid two-character lowercase ISO-639 language code and if present the optional <CC> is a valid two-character uppercase ISO-3166 country code. Both '_' and '-' are supported as delimiters. • `parentNodeId` - Specifies the DOM identifier of the node to which the form is added. The value cannot be `null`. If you are using an `iframe`, you can pass the ID of the `iframe` element on the page. For more information, see iframe Integration. • `onSuccess` - A function that is called once the form is successfully initialized. The Form object is passed into this function. This function can be used to add custom callback handlers that implement lifecycle events such as `submit`, `close`, and `cancel`. • `onError` - A function that is called if any errors are encountered in initializing the form. The function will receive any exception that was encountered during the initialization. • `JSONP` - Informs the Forms Runtime Adapter to use JSON with Padding (JSONP) when loading JSON resources. The default value is `false`. When the custom client and Forms Runtime Adapter are hosted on different servers, set the JSONP parameter as `true`. In this scenario, there are SOP (Single Origin Policy) issues while loading JSON resources. By using the JSONP technique, the JSON response is wrapped to a call to a function by the server and sent to the client. A JSON resource can then be loaded using a script tag to avoid any SOP violations.

Method	Description
`loadFormWithRemoteData(String url, String initDataURL, String bomJSPath, String locale, String parentNodeId, Function onSuccess, Function onError, Boolean JSONP)`	Loads the form at the specified URL. The parameter details are as follows: • `url` - Specifies the URL to the form JSON representation, e.g. "`http://<hostname:port>/bpmresources/com.example.myproject_1.0.0.20201105175821853/wp/openspaceGWTPull_DefaultChannel/openspaceGWTPull_DefaultChannel/FindAddress/GetAddress/Getuserdetails/Getuserdetails.gwt.json`". • `initDataURL` - Specifies the URL to the form initial data. • `bomJSPath` - Specifies the root folder path used for loading the BOM JavaScript files used by the form, e.g. "`http://<hostname:port>/bpmresources`" • `locale` - Specifies the locale to be used in the form runtime with format <lc>[_<CC>] where [] denotes optionality, e.g. "en_US". The locale need to be represented such that <lc> is a valid two-character lowercase ISO-639 language code and if present the optional <CC> is a valid two-character uppercase ISO-3166 country code. Both '_' and '-' are supported as delimiters. • `parentNodeId` - Specifies the DOM identifier of the node to which the form should be added. The value cannot be `null`. If you are using an `iframe`, you can pass the ID of the `iframe` element on the page. For more information, see iframe Integration. • `onSuccess` - A function that is called once the form is successfully initialized. The Form object is passed into this function. This can be used to add custom callback handlers that implement lifecycle events such as `submit`, `close`, and `cancel`. • `onError` - A function that is called if any errors are encountered in initializing the form. The function will receive any exception that was encountered during the initialization.

Method	Description
	• `JSONP` - Informs the Forms Runtime Adapter to use JSON with Padding (JSONP) when loading JSON resources. The default value is `false`. When the custom client and Forms Runtime Adapter are hosted on different servers, set the JSONP parameter as `true`. In this scenario, there are SOP (Single Origin Policy) issues while loading JSON resources. By using the JSONP technique, the JSON response is wrapped to a call to a function by the server and sent to the client. A JSON resource can then be loaded using a script tag to avoid any SOP violations.
`renderStaticView()`	Renders a tree representation of the data provided in the `InitialData` parameter. The API has the following parameters: • `initialData`: specifies the JSON representation of the initial data that are provided to the form. • `bomJSPath`: specifies the root folder path used for loading the BOM JavaScript files used by the form. For example: `http://<hostname:port>/bpmresources` • `locale`: specifies the locale to be used in the runtime form with the `<lc>[_<CC>]` format, where `[]` denotes optionality. For example: "*en_US*". The locale needs to be represented such that `<lc>` is a valid two-character lowercase ISO-639 language code, and, if present, the optional `<CC>` is a valid two-character uppercase ISO-3166 country code. Both '_' and '–' are supported as delimiters. • `parentNodeId`: specifies the DOM identifier of the node to which the form must be added. The value cannot be null. If you are using an `iframe`, you can pass the ID of the `iframe` element on the page. For more information, see iframe Integration. • `onSuccess`: is a function that is called after the form is successfully initialized. The Form object is passed into this

Method	Description
	function. It can be used to add custom callback handlers that implement lifecycle events such as submit, close, and cancel. • `onError`: is a function that is called on encountering any error in initializing the form. The function receives the exception encountered during the initialization. • `provideCloseAction`: if it is set to `true`, the form is rendered with a button that closes the form and cleans up any resources used. If it is set to `false`, then it is the responsibility of the containing application to clean up the form when it is no longer needed. • `JSONP`: informs the Forms Runtime Adapter to use JSON with Padding (JSONP) when loading JSON resources. The default value is `false`. When the custom client and Forms Runtime Adapter are hosted on different servers, set the JSONP parameter to `true`. In this scenario, there are SOP (Single Origin Policy) issues while loading JSON resources. By using the JSONP technique, the JSON response is wrapped as a function by the server and is sent to the client. A JSON resource can then be loaded using a script tag to avoid any SOP violations.

Note: For the list of language codes and country codes required for specifying the `locale` parameter, visit the following websites:

- List of language codes - http://www.loc.gov/standards/iso639-2/langhome.html
- List of country codes - http://www.iso.org/iso/country_codes/iso_3166_code_lists.html

The details of the `logger()` method are as follows:

com.tibco.forms.client.FormRunner.logger Class

Method	Return Value	Description
`fatal(String message)`	Void	Logs the given messages at the `fatal` logging level.
`error(String message)`	Void	Logs the given messages at the `error` logging level.
`warn(String message)`	Void	Logs the given messages at the `warn` logging level.
`info(String message)`	Void	Logs the given messages at the `info` logging level.
`debug(String message)`	Void	Logs the given messages at the `debug` logging level.
`trace(String message)`	Void	Logs the given messages at the `trace` logging level.
`isFatalEnabled()`	Boolean	Checks whether the `Fatal` logging level is enabled. It returns `true` if the logging level is enabled, and `false` otherwise.
`isErrorEnabled()`	Boolean	Checks whether the `Error` logging level is enabled. It returns `true` if the logging level is enabled, and `false` otherwise.
`isWarnEnabled()`	Boolean	Checks whether the `Warn` logging level is enabled. It returns `true` if the logging level is enabled, and `false` otherwise.

Method	Return Value	Description
`isInfoEnabled()`	Boolean	Checks whether the `Info` logging level is enabled. It returns `true` if the logging level is enabled, and `false` otherwise.
`isDebugEnabled()`	Boolean	Checks whether the `Debug` logging level is enabled. It returns `true` if the logging level is enabled, and `false` otherwise.
`isTraceEnabled()`	Boolean	Checks whether the `Trace` logging level is enabled. It returns `true` if the logging level is enabled, and `false` otherwise.

com.tibco.forms.client.Form

The `com.tibco.forms.client.Form` class provides access to the runtime form object. This object enables you to access panes and controls within the form, register handlers for form actions, and access data to be submitted back to the server.

The `com.tibco.forms.client.Form` class has six fields that are used for setting action handlers. It also implements the methods listed in the table below.

com.tibco.forms.client.Form Class - Field Details

Field	Data Type	Description
`ACTION_APPLY`	String	Identifies the "apply" action.
`ACTION_CANCEL`	String	Identifies the "cancel" action.
`ACTION_CLOSE`	String	Identifies the "close" action.

Field	Data Type	Description
`ACTION_RESET`	String	Identifies the "reset" action.
`ACTION_SUBMIT`	String	Identifies the "submit" action.
`ACTION_VALIDATE`	String	Identifies the "validate" action.

com.tibco.forms.client.Form Class - Method Details

Method	Return Value	Description
`destroy()`	Void	Removes the form from its container and also releases its resources. This can be called by the client application to close the form.
`getFactory()`	Object	Returns the factory object for the given form. This provides access to the BOM JavaScript factories associated with the BOM types used by this form, as documented under the *factory* variable available within form action and validation scripts. **Note:** This method is not supported when an `iframe` is used to load the form using the built-in `iframe` integration support.
`getLoadStats()`	LoadStat	Returns an array of LoadStat objects. Each statistic represents the measurement of a particular phase of the form load. This can be used by

Method	Return Value	Description

		applications to report this information in the user interface or otherwise log the information.
		
		To enable collecting load statistics at runtime, the URL used by the client application to load the form should contain the parameter `tibco_instr` with a value `true`. Otherwise `getLoadStats()` method would return an empty array.
		
		This method takes an optional callback function as an argument to support `iframe` integration, where the method returns asynchronously:
		
		`getLoadStats(Function callback)`
		
		The callback function is optional for `non-iframe` mode, but required for `iframe` mode.
		
		The method still returns the value in `non-iframe` mode for backward compatibility. However, the use of the callback function is recommended for both `iframe` and `non-iframe` modes. Example:
		
		`form.getLoadStats(function (loadStats){ for (idx in loadStats) alert(loadStats [idx].label + ": " + (loadStats [idx].endTime - loadStats [idx].startTime) + " ms"; });`
		
		To use the `getLoadStats()` method, the

Method	Return Value	Description
		client application needs to subscribe to the TIBCO PageBus event '`com.tibco.forms.form.loaded`'. See com.tibco.forms.client.LoadStat for more details.
`getPackage()`	Object	Returns the object that provides access to the BOM JavaScript package definitions associated with the BOM types used by this form, as documented under the *pkg* variable available within form action and validation scripts. **Note:** This method is not supported when an `iframe` is used to load the form using the built-in `iframe` integration support.
`getResource()`	Object	Returns the object that provides access to the resource bundles associated with this form, as documented under the *resource* variable available within form action and validation scripts. **Note:** This method is not supported when an `iframe` is used to load the form using the built-in `iframe` integration support.
`getSerializedParameters()`	String	Returns a JSON representation of the data being managed by the form. This is typically called from a submit handler to send the final results back to the server. This method takes an optional callback function as an argument to support

Method	Return Value	Description

		`iframe` integration, where the method returns asynchronously:
		`getSerializedParameters (Function callback)`
		The callback function is optional for `non-iframe` mode but required for `iframe` mode.
		The method still returns the value in `non-iframe` mode for backward compatibility. However, use of the callback function is recommended for both `iframe` and `non-iframe` modes. Example:
		`form.getSerializedParameters (function(data) { alert('form data: ' + data); });`

`setActionHandler( String actionName, Function handler)`	Void	Adds a handler to the form that is invoked when the specified action is invoked in the form. Note that any handler already registered for this action will be replaced by this handler. The parameter details are as follows: • `actionName` - Used to specify the name of the • action (For example, `ACTION_CLOSE`). If the action is `ACTION_SUBMIT`, then all the validations in the form will be invoked before invoking the callback handlers. If any of the validations fail, then the

Method	Return Value	Description
		callback handler will not be invoked. • `handler` - This is the function that is invoked for the specified `actionName`. The form may have only one handler for each action. If this method is called more than once for the same `actionName`, the handler set previously will be replaced. Passing in `null` for this parameter will remove the handler for this particular action. The method signature of the `handler` function has two arguments: a string for the `actionName` and the form object.

Properties	Return Value	Description
locale	String	Returns the string representation of the locale being used to render the form. This method takes an optional callback function as an argument to support `iframe` integration, where the method returns asynchronously: `locale(Function callback)` The callback function is optional for `non-iframe` mode, but required for `iframe` mode. The method still returns the value in `non-iframe` mode for backward compatibility. However, the use of the callback function is recommended for both `iframe` and `non-iframe`

Properties	Return Value	Description

		modes. Example:
		`form.locale(function(locale) {`
		`  alert("Form Locale is: " + locale);`
		`});`

com.tibco.forms.client.LoadStat

The `com.tibco.forms.client.LoadStat` class helps you to measure the load-time performance of the form. Each statistic has three fields: a description, a start time, and an end time. The times are measured in milliseconds from when the form load began.

To use `com.tibco.forms.client.LoadStat`, the form loading has to be complete, including data loading and invocation of form open rules. The TIBCO PageBus event '`com.tibco.forms.form.loaded`' is published when the form loading is complete. The client application needs to subscribe to this event in order to be notified that the form loading is complete and then use the `getLoadStats()` method.

Example:

```
var callback = function(subject, form) {
    var stat = form.getLoadStats();
    for (var i=0; i<stat.length; i++) {
        alert(stat[i].label+" |                End Time : "
            + stat[i].endTime+" |              Begin Time : "
            + stat[i].startTime);
    }
}
PageBus.subscribe('com.tibco.forms.form.loaded', null,callback);
```

The details of the fields are as follows:

com.tibco.forms.client.LoadStat Class - Field Details

Field	Data Type	Description
`label`	String	Describes what is being measured by this particular loadstat.
`startTime`	Number	The time, in milliseconds, when this particular measurement phase began. This time is relative to the instant when the form load began.
`endTime`	Number	The time, in milliseconds, when this particular measurement phase ended. This time is relative to the instant when the form load began.

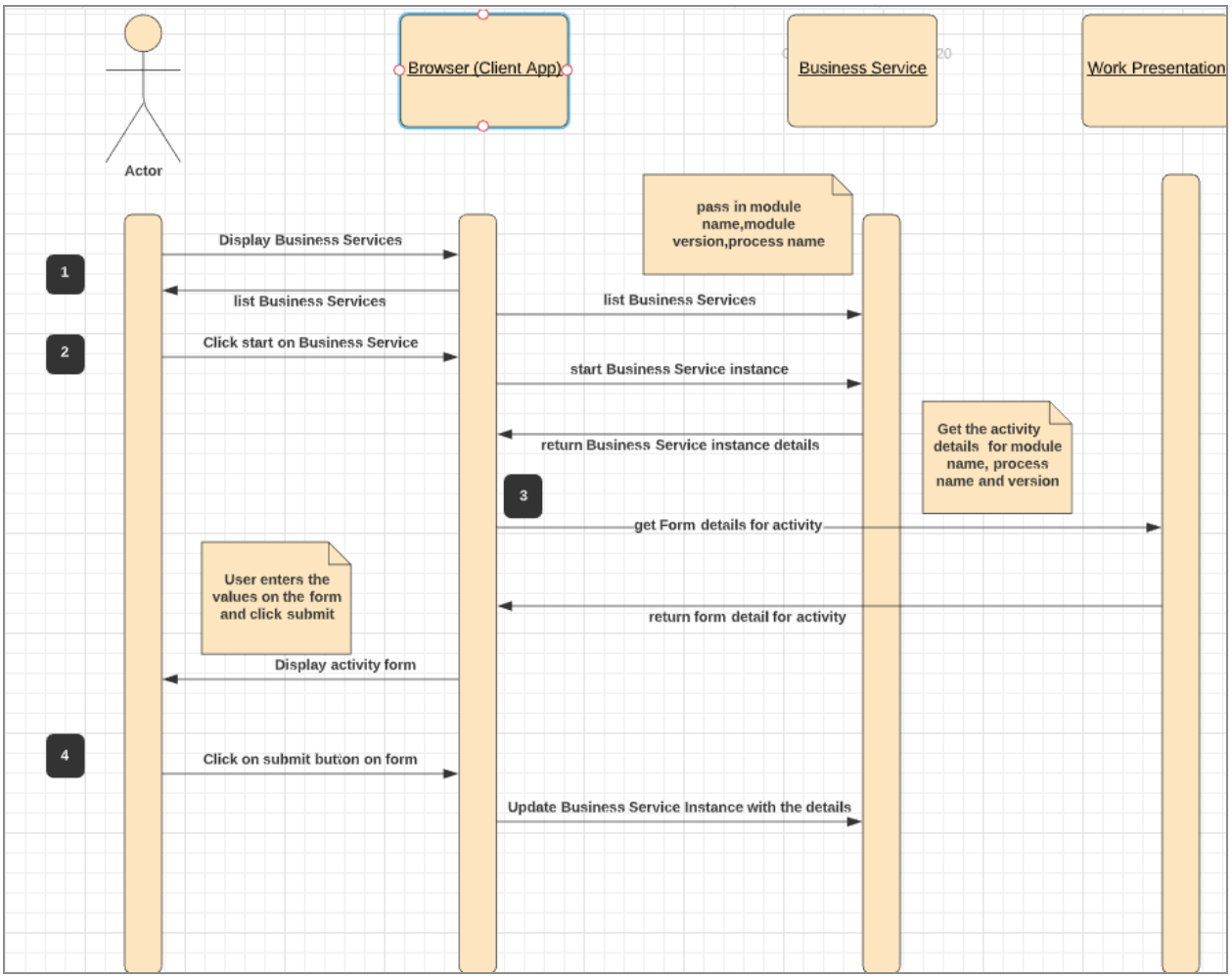
Working With Business Services

Starting a business service involves listing all the available business services and then selecting a business service to start.

The following sub-topic describes the web service operation that allows you to work with business services.

Starting a Business Service

This topic provides an example that shows how calls to the Work Manager Services API can be used to start and update a business service.



Note: The following step-by-step description corresponds to the process shown in the diagram. The descriptions are from the perspective of the REST API. The process of performing operations using the REST API is explained with the help of an example.

Procedure

1. When you login to the client application and click **Business Services**, GET business services rest API is called passing in the following filter string. For more details, refer to **API Explorer**.
2. When you click **Start** on Business Service, the POST call of call instance is called. POST call of call instance passes the following parameters:
   - `module name`

- `process name`
- `version number`
- `/Instances` is the moduleName name of module/package to query on.
- moduleVersion version of module/package to query on

```
{
   "moduleName": "string",
   "processName": "string",
   "moduleVersion": "string",
   "data": "string"
}
```

- The response returns the execution state, instance ID, activity information, and the payload.

```
{
  "instanceId": "123",
  "moduleName": "/SampleProcesses/Sample
Tests/SampleTests.xpdl",
  "moduleVersion": "1.0.0.20201106103914259",
  "processName": "SampleBusinessService",
  "state": {
    "state": "ACTIVE",
    "failedMessage": null
  },
  "activityInfo": {
    "activityId": "abc",
    "activityName": "UserTask",
    "activityPageflowName": "StartEvent",
    "activityData": "{\"body\":{}}",
    "activityProcessModuleName": "/SampleProcesses/Sample
Tests/SampleTests.xpdl",
    "activityProcessModuleVersion": "1.0.0.20201106103914259"
  }
}
```

3. If Business Service has a form associated with it, Work presentation **WorkitemPageFlow** GET rest API is called by passing the module name, version, and process name that is returned in the earlier call.
    - `/WorkItemPageFlow` is the Pageflow associated with the work item.

- `processName`: the name of the Process Template to query on.
- `moduleVersion`: the version of the Package to query on.
- The response returns form details for the activities of the process associated with the process name that is passed as part of the request. Form identifier points to the form that represents the user task activity in the business service. The form is rendered by the client for the user to enter the following details:

```
[
  {
    "moduleName":  "/SampleProcesses/Sample
Tests/SampleTests.xpdl",
    "processName": "SampleBusinessService",
    "moduleVersion": "1.0.0.20201106103914259",
    "pageflowActivities": [
      {
        "activityName": "UserTask",
        "activityId": "_r_2wp49sEeq3q_CEUhibZA",
        "formIdentifier": "com.example.sampleprocesses_
1.0.0.20201106103914259/wp/openspaceGWTPull_
DefaultChannel/.default/SanityTests/GetandViewDataStartEvent/Us
erTask/UserTask.gwt.json"
      }
    ]
  }
]
```

4. After you enter the details and click the submit button of the form, PUT call of instance is called. The data received from the Form is updated and the data is submitted back to the server.
   - `/Instances` Updates a specified Process Instance.

```
{
  "instanceId": "123",
  "moduleName": "/SampleProcesses/Sample
Tests/SampleTests.xpdl",
  "moduleVersion": "1.0.0.20201106103914259",
  "processName": "SampleBusinessService",
  "state": {
    "state": "ACTIVE",
    "failedMessage": null
  },
```

```
  "activityInfo": {
    "activityId": "abc",
    "activityName": "UserTask",
    "activityPageflowName": "StartEvent",
    "activityData": "{\"body\":{}}",
    "activityProcessModuleName": "/SampleProcesses/Sample
Tests/SampleTests.xpdl",
    "activityProcessModuleVersion": "1.0.0.20201106103914259"
  }
```

List Business Services

Lists all available business services.

Endpoint	`http://<hostname>/bpm/pageflow/v1/businessServices?$filter=targetDevice eq 'Desktop'`
	Request Method:
	`GET`

Request Payload	N/A

Response	<pre>[
	    {
	        "processId": "131",
	        "processName": "Make_Loan_Application",
	        "processLabel": "Make_Loan_Application",
	        "processExtName":
	"processOut/pageflow/myDemo.xpdl/Make_Loan_Application.bpel",
	        "processType": "BUSINESSSERVICE",
	        "channelIds": [
	            "openspaceGWTPull_DefaultChannel"
	        ],
	        "category": "myDemo/myDemo",
	        "privileges": [],
	        "moduleId": "84",
	        "moduleName": "/myDemo/Process Packages/myDemo.xpdl",
	        "moduleInternalName": "/myDemo/Process
	Packages/myDemo.xpdl",</pre>

```
        "moduleVersion": "1.0.0.20240212174759102",
        "hasInputParameters": false,
        "targetDevice": "Desktop"
    }
]
```

Start Business Service

Creates an instance of a business service.

Endpoint	`http://<hostname>/bpm/pageflow/v1/instances` Request Method: POST
Request Payload	`{ "moduleName": "/myDemo/Process Packages/myDemo.xpdl", "processName": "Make_Loan_Application", "moduleVersion": "1.0.0.20240212174759102" }`
Response	`{ "instanceId": "p:0a20co", "moduleName": "/myDemo/Process Packages/myDemo.xpdl", "moduleVersion": "1.0.0.20240212174759102", "processName": "Make_Loan_Application", "state": { "state": "ACTIVE", "failedMessage": null }, "activityInfo": { "activityId": "t:002gco.3", "activityName": "UserTask", "activityProcessPackageId": null,`

```
        "activityPageflowName": "Make_Loan_
Application",
        "activityData": "{\"body\":{}}",
        "activityProcessModuleName":
"/myDemo/Process Packages/myDemo.xpdl",
        "activityProcessModuleVersion":
"1.0.0.20240212174759102"
    }
}
```

Update Business Service

Updates a specified business service instance.

Endpoint

```
http://<hostname>/bpm/pageflow/v1/instances
Request Method:
PUT
```

Request Payload

```
{
  "instanceId": "p:0a20cp",
  "moduleName": "/myDemo/Process Packages/myDemo.xpdl",
  "moduleVersion": "1.0.0.20240212174759102",
  "processName": "Make_Loan_Application",
  "state": {
    "state": "ACTIVE",
    "failedMessage": null
  },
  "activityInfo": {
    "activityId": "t:002gcp.3",
    "activityName": "UserTask",
    "activityProcessPackageId": null,
    "activityPageflowName": "Make_Loan_Application",
    "activityData": "{\"body\":{\"inouts\":
[{\"name\":\"ApplicantDetails\", \"structured\":
[{\"name\":\"Fred\", \"income\":123,
\"emailAddress\":\"fred@fred.com\", \"address\":
{\"line1\":\"green street\", \"town\":\"blue town\"}}]},
{\"name\":\"LoanAmount\", \"simple\":[333]}], \"outputs\":
```

```
[]}}",
    "activityProcessModuleName": "/myDemo/Process
Packages/myDemo.xpdl",
    "activityProcessModuleVersion": "1.0.0.20240212174759102"
  }
}
```

Response

```
{
    "instanceId": "p:0a20cp",
    "moduleName": "/myDemo/Process Packages/myDemo.xpdl",
    "moduleVersion": "1.0.0.20240212174759102",
    "processName": "Make_Loan_Application",
    "state": {
        "state": "COMPLETED",
        "failedMessage": null
    },
    "activityInfo": {
        "activityId": "t:002gcp.3",
        "activityName": "UserTask",
        "activityProcessPackageId": null,
        "activityPageflowName": "Make_Loan_Application",
        "activityData": "{\"body\":{\"inouts\":
[{\"name\":\"LoanAmount\",\"simple\":[\"333\"]},
{\"name\":\"ApplicantDetails\",\"structured\":
[{\"income\":123,\"emailAddress\":\"fred@fred.com\",\"addres
s\":{\"town\":\"blue town\",\"line1\":\"green
street\"},\"name\":\"Fred\"}]},
{\"name\":\"LoggedInUser\",\"simple\":[\"tibco-admin\"]}]}}",
        "activityProcessModuleName": "/myDemo/Process
Packages/myDemo.xpdl",
        "activityProcessModuleVersion":
"1.0.0.20240212174759102"
    },
    "data": "{\"body\":{\"inouts\":
[{\"name\":\"LoanAmount\",\"simple\":[\"333\"]},
{\"name\":\"ApplicantDetails\",\"structured\":
[{\"income\":123,\"emailAddress\":\"fred@fred.com\",\"addres
s\":{\"town\":\"blue town\",\"line1\":\"green
street\"},\"name\":\"Fred\"}]},
{\"name\":\"LoggedInUser\",\"simple\":[\"tibco-admin\"]}]}}"
}
```

Cancel Business Service

Deletes a specified business service instance.

Endpoint	`http://<hostname>/bpm/pageflow/v1/instances/p:0a20ct`
	`Request Method:`
	`DELETE`

Request Payload	N/A

Response	<pre>{
	    "instanceId": "p:0a20ct",
	    "moduleName": "/myDemo/Process Packages/myDemo.xpdl",
	    "moduleVersion": "1.0.0.20240212174759102",
	    "processName": "Make_Loan_Application",
	    "state": {
	        "state": "CANCELLED",
	        "failedMessage": null
	    }
	}</pre>

Working With Case Data

The following sub-topics describe additional details regarding case data.

Query and Fetch Case Data Types

TIBCO BPM Enterprise allows you to determine the JSON schema of a case type in the following ways:

- For more details, go to API Explorer > Case Data Management Service, and GET operations for the case types.

  API Explorer - http://<*host:port*>/apps/api-explorer/index.html#!/home

- Alternatively, you can use the following GET query.

  Get Case Types - http://<*host:port*>/bpm/case/v1/types?$top=100&$filter=isCase%20eq%20TRUE

  This returns a list of all the case data types present in the system.

Each data type lists the following details:

- ApplicationMajorVersion, applicationId, name, namespace, label
- Data attributes with metadata like identifier, mandatory, searchable, summary, length constraints, default values
- Summary attributes
- Case states with label, value and isTerminal (only for terminal states)

Sample Response:

```
[
{
  "name": "AdditionalOrder",
  "label": "AdditionalOrder",
  "isCase": true,
  "namespace": "com.example.samplebdsproject1",
  "applicationId": "com.example.samplebdsproject1",
```

```
    "applicationMajorVersion": 1,
    "attributes": [
      {
        "name": "additionalOrderID",
        "label": "AdditionalOrderID",
        "type": "Text",
        "isIdentifier": true,
        "isAutoIdentifier": true,
        "isMandatory": true,
        "isSearchable": true,
        "isSummary": true
      },
      {
        "name": "caseState1",
        "label": "caseState1",
        "type": "Text",
        "isState": true,
        "isMandatory": true,
        "isSearchable": true,
        "isSummary": true
      }
    ],
    "summaryAttributes": [
      {
        "name": "additionalOrderID",
        "label": "AdditionalOrderID",
        "type": "Text",
        "isIdentifier": true,
        "isAutoIdentifier": true,
        "isMandatory": true,
        "isSearchable": true,
        "isSummary": true
      },
      {
        "name": "caseState1",
        "label": "caseState1",
        "type": "Text",
        "isState": true,
        "isMandatory": true,
        "isSearchable": true,
        "isSummary": true
      }
    ],
    "states": [
      {
        "label": "Picked",
        "value": "PICKED"
```

```
      },
      {
        "label": "Packed",
        "value": "PACKED"
      },
      {
        "label": "Delivered",
        "value": "DELIVERED",
        "isTerminal": true
      }
    ],
    "links": [
      {
        "name": "order",
        "label": "Order",
        "type": "Order"
      }
    ]
  },
  {
    "name": "Order",
    "label": "Order",
    "isCase": true,
    "namespace": "com.example.samplebdsproject1",
    "applicationId": "com.example.samplebdsproject1",
    "applicationMajorVersion": 1,
    "attributes": [
      {
        "name": "orderID",
        "label": "OrderID",
        "type": "Text",
        "isIdentifier": true,
        "isMandatory": true,
        "isSearchable": true,
        "isSummary": true,
        "constraints": {
          "length": 50
        }
      },
      {
        "name": "orderState",
        "label": "OrderState",
        "type": "Text",
        "isState": true,
        "isMandatory": true,
        "isSearchable": true,
        "isSummary": true
```

```
    },
    {
      "name": "name",
      "label": "Name",
      "type": "Text",
      "isSearchable": true,
      "constraints": {
        "length": 50
      }
    },
    {
      "name": "quantity",
      "label": "Quantity",
      "type": "Number"
    },
    {
      "name": "product",
      "label": "Product",
      "type": "Text",
      "isSearchable": true,
      "constraints": {
        "length": 50
      }
    }
  ],
  "summaryAttributes": [
    {
      "name": "orderID",
      "label": "OrderID",
      "type": "Text",
      "isIdentifier": true,
      "isMandatory": true,
      "isSearchable": true,
      "isSummary": true,
      "constraints": {
        "length": 50
      }
    },
    {
      "name": "orderState",
      "label": "OrderState",
      "type": "Text",
      "isState": true,
      "isMandatory": true,
      "isSearchable": true,
      "isSummary": true
    }
```

```
    ],
    "states": [
      {
        "label": "Picked",
        "value": "PICKED"
      },
      {
        "label": "Packed",
        "value": "PACKED"
      },
      {
        "label": "Delivered",
        "value": "DELIVERED",
        "isTerminal": true
      }
    ],
    "links": [
      {
        "name": "additionalOrder",
        "label": "AdditionalOrder",
        "type": "AdditionalOrder"
      }
    ]
  }
]
```

List Case types

Lists all the case data types that match the specified query parameters.

Endpoint	`http://<hostname>/bpm/case/v1/types?$top=100&$filter=isCase%20eq%20TRUE` Request Method: GET
Request Payload	N/A

Response


```
[
    {
        "name": "LoanApplication",
        "label": "Loan Application",
        "isCase": true,
        "namespace": "com.example.p2p",
        "applicationId": "com.example.p2p",
        "applicationMajorVersion": 1,
        "attributes": [
            {
                "name": "loanapplicationId",
                "label": "Loan application Id",
                "type": "Text",
                "isIdentifier": true,
                "isAutoIdentifier": true,
                "isMandatory": true,
                "isSearchable": true,
                "isSummary": true
            },
            {
                "name": "applicantdetails",
                "label": "Applicant details",
                "type": "Applicantdetails",
                "isStructuredType": true,
                "isMandatory": true
            },
            {
                "name": "loanAmount",
                "label": "Loan Amount",
                "type": "FixedPointNumber",
                "constraints": {
                    "length": 10,
                    "decimalPlaces": 0
                }
            },
            {
                "name": "creditScore",
                "label": "Credit Score",
                "type": "FixedPointNumber",
```

```
            "constraints": {
                "length": 10,
                "decimalPlaces": 0
            }
        },
        {
            "name": "loanoffers",
            "label": "Loan Offers",
            "type": "LoanOffer",
            "isStructuredType": true,
            "isArray": true
        },
        {
            "name": "acceptedoffer",
            "label": "Accepted Offer",
            "type": "LoanOffer",
            "isStructuredType": true
        },
        {
            "name": "applicationDate",
            "label": "Application Date",
            "type": "Date"
        },
        {
            "name": "applicationAccepted",
            "label": "Application Accepted",
            "type": "Boolean"
        },
        {
            "name": "rejectedReason",
            "label": "Rejected Reason",
            "type": "Text",
            "constraints": {
                "length": 50
            }
        },
        {
            "name": "applicationState",
            "label": "Application State",
            "type": "Text",
            "isState": true,
```

```
el": "Offers Available",
                "value": "OFFERSAVAILABLE"
            },
            {
                "label": "Loan Accepted",
                "value": "LOANACCEPTED",
                "isTerminal": true
            },
            {
                "label": "Loan Rejected",
                "value": "LOANREJECTED",
                "isTerminal": true
            }
        ]
    }
]
```

List Cases for Case Type

Returns all the Types that match the specified query parameters.

Endpoint	`http://<hostname>/bpm/case/v1/cases?$top=100&$filter=caseType eq 'com.example.p2p.LoanApplication' and applicationMajorVersion eq 1 and isInTerminalState eq FALSE Request Method: GET`
Request Payload	N/A
Response	`[{ "caseReference": "27-com.example.p2p.LoanApplication-1-6", "casedata": "{\"loanAmount\": 444, \"loanoffers\": [{}, {}, {}, {\"amount\": 444}, {\"amount\": 11, \"lender\": \"sdf\", \"duration\": 1, \"interestRate\": 123}], \"creditScore\": 450, \"applicationDate\": \"2024-02-12\", \"applicantdetails\":`

```
{\"name\": \"Jack\", \"income\": 111, \"address\": {\"line1\":
\"marg\"}}, \"applicationState\": \"OFFERSAVAILABLE\",
\"loanapplicationId\": \"2\", \"applicationAccepted\": true}",
        "summary": "
{\"loanapplicationId\":\"2\",\"applicationState\":\"OFFERSAVAIL
ABLE\"}",
        "metadata": {
            "createdBy": "tibco-admin",
            "creationTimestamp": "2024-02-12T18:34:56.063Z",
            "modifiedBy": "tibco-admin",
            "modificationTimestamp": "2024-02-12T18:39:16.746Z"
        }
    }
]
```

List Case Actions

Lists all available Case Actions.

Endpoint	`http://<hostname>/bpm/pageflow/v1/caseActions?$filter=caseRef eq '27-com.example.p2p.LoanApplication-1-6' and caseState eq 'OFFERSAVAILABLE'`
	Request Method:
	`GET`

Request Payload	N/A
Response	`[ { "processId": "133", "processName": "AcceptOffer", "processLabel": "Accept Offer", "processExtName": "processOut/pageflow/myDemo.xpdl/AcceptOffer.bpel",`

```
        "processType": "CASEACTION",
        "channelIds": [
            "openspaceGWTPull_DefaultChannel"
        ],
        "privileges": [],
        "caseStates": [
            "OFFERSAVAILABLE"
        ],
        "caseInfo": {
            "caseClassName":
"com.example.p2p.LoanApplication",
            "caseVersion": "1",
            "caseRefParamName": "LoanApplicationRef",
            "caseStatePropertymName": "applicationState"
        },
        "moduleId": "84",
        "moduleName": "/myDemo/Process Packages/myDemo.xpdl",
        "moduleInternalName": "/myDemo/Process
Packages/myDemo.xpdl",
        "moduleVersion": "1.0.0.20240212174759102"
    },
    {
        "processId": "132",
        "processName": "RejectOffer",
        "processLabel": "Reject Offer",
        "processExtName":
"processOut/pageflow/myDemo.xpdl/RejectOffer.bpel",
        "processType": "CASEACTION",
        "channelIds": [
            "openspaceGWTPull_DefaultChannel"
        ],
        "privileges": [],
        "caseStates": [
            "OFFERSAVAILABLE"
        ],
        "caseInfo": {
            "caseClassName":
"com.example.p2p.LoanApplication",
            "caseVersion": "1",
            "caseRefParamName": "LoanApplicationRef",
            "caseStatePropertymName": "applicationState"
        },
        "moduleId": "84",
        "moduleName": "/myDemo/Process Packages/myDemo.xpdl",
```

Read Cases

Returns all Cases that match the specified query parameters.

Endpoint	`http://<hostname>/bpm/case/v1/cases/27-com.example.p2p.LoanApplication-1-6?$select=cr,c`
	Request Method:
	`GET`

Request Payload	N/A
Response	`{ "caseReference": "27-com.example.p2p.LoanApplication-1-6", "casedata": "{\"loanAmount\": 444, \"loanoffers\": [{}, {}, {}, {\"amount\": 444}, {\"amount\": 11, \"lender\": \"sdf\", \"duration\": 1, \"interestRate\": 123}], \"creditScore\": 450, \"applicationDate\": \"2024-02-12\", \"applicantdetails\": {\"name\": \"Jack\", \"income\": 111, \"address\": {\"line1\": \"marg\"}}, \"applicationState\": \"OFFERSAVAILABLE\", \"loanapplicationId\": \"2\", \"applicationAccepted\": true}" }`

Conventional Database View of a Case Type

In TIBCO BPM Enterprise, it is not always necessary to create a database view of a case type. A database view named "cdm_cases" already exists for the case manager. This lists

the cases for different case types created in the TIBCO BPM Enterprise.

The cdm_cases view has the following columns:

Columns	Description
case_identifier	an identifier for a given case
casereference	reference for the case with version
unversioned_casereference	reference for the case without version
type	case type for a given case
version	version for a given case
state	current state for a given case
casedata	data associated with the case
is_active	whether the case type is active
creation_timestamp	time of creation of the case
modification_timestamp	time of modification of the case
completed_case_duration	total time elapsed from the start of the case to its completion
application_name	name or label for the deployed application
application_id	unique id for the deployed application
application_version	the version of the deployed application

Working with Process Manager

The TIBCO® BPM Enterprise Process Management Service is used to manage process templates and process instances.

Creating an Instance from a Specified Process Template

When you deploy a project and you start a process for it, you can start it from the **Process Manager > Process Templates** tab by clicking the **Start** button. However, if the project has some parameters to be passed, you can only start it directly from the API or else it must be started through a business service.

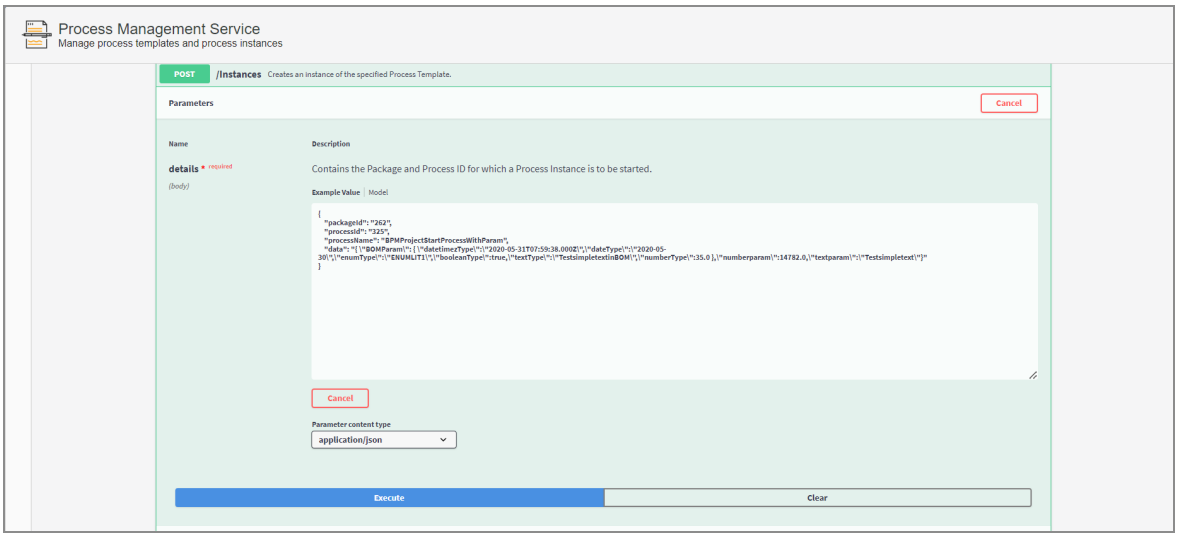
Let us consider an example of a BPM project which needs some parameters to be passed for the process instance to be created.

The sample for data parameter is as follows:

```
"data": "{ \"BOMParam\":
{ \"datetimezType\":\"2020-05-31T07:59:38.000Z\",\"dateType\":\"2020-05-
30\",\"enumType\":\"ENUMLIT1\",\"booleanType\":true,\"textType\"
:\"TestsimpletextinBOM\",\"numberType\":35.0}
```

Procedure

1. Import the BPM project which needs some parameters to be passed for the process instance to be created.
2. In the Process Management Service, enter sample API where the sample REST call URL is: `http://<hostname>/bpm/processes/v1/instances`



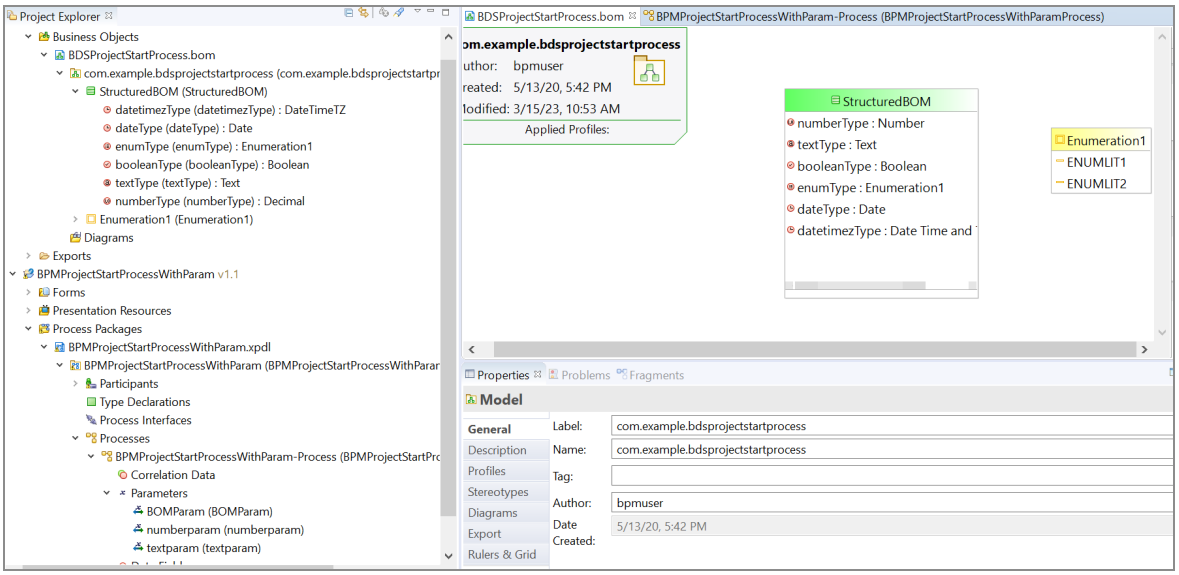
The sample data parameter is as follows:

```
"data": "{ \"BOMParam\":
{ \"datetimezType\":\"2020-05-
31T07:59:38.000Z\",\"dateType\":\"2020-05-
30\",\"enumType\":\"ENUMLIT1\",\"booleanType\":true,\"textType\"
:\"TestsimpletextinBOM\",\"numberType\":35.0}
```

The sample for how to pass a simple type of parameter is as follows:

```
,\"numberparam\":14782.0,\"textparam\":\"Testsimpletext\"}"
```

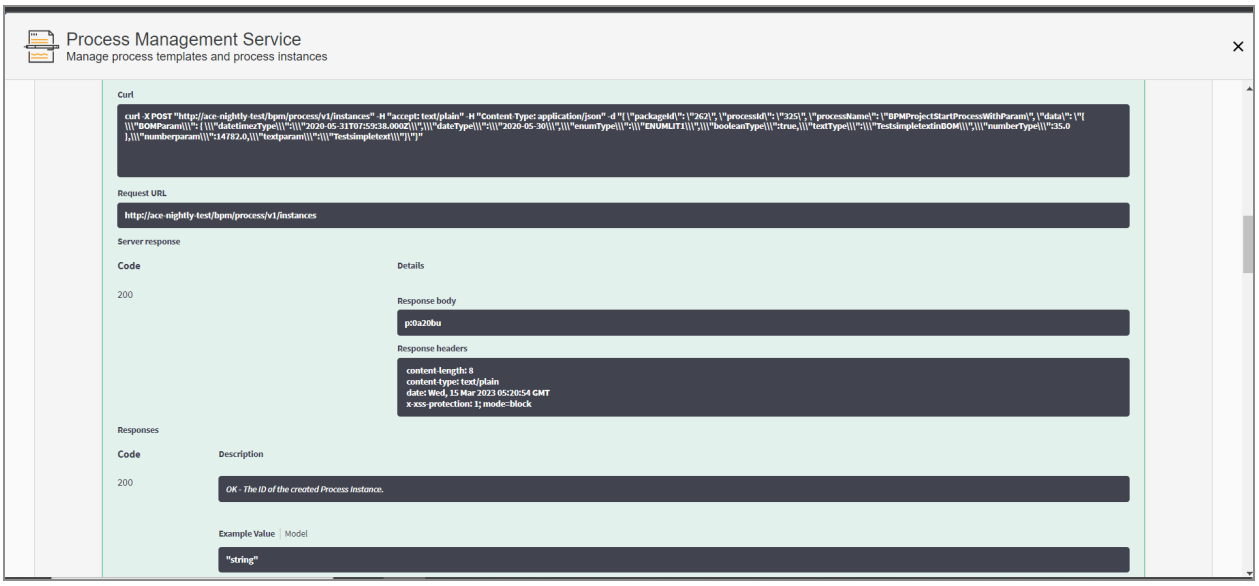
The structure of **BOMParam** is as follows:



3. Click **Execute**.

Result

The Process Instance starts.



Example:

Endpoint	`http://ace-nightly-test/bpm/process/v1/instancesRequest`

	`Method:` `POST`

Request Payload	`{`
	`"packageId": "1",`
	`"processId": "1"`
	`}`

Response	`p:0a203n`

Sending an Event to a Process

The /SendEvent API from the API explorer is used to send an event to either a specific process instance, which must exist at the time when the API is called by using the `instanceId` attribute, or through correlation data, using the `correlationData` attribute. You should supply either one of these attributes, not both.

A data payload can be sent to the process to update its data fields using the `data` attribute, for which the format is similar to creating any instance.

If you are using correlation data, the instance you are targeting does not need to exist at the time you call the API, but if multiple correlation data fields are used, the order in which they are specified in the API must match the order in which they are specified on the task in TIBCO Business Studio - BPM Edition.

Example:

Endpoint	`http://ace-nightly-test.emea.tibco.com/bpm/process/v1/sendEvent`
	`Request Method:`
	`Post`

Request Payload	{
	  "instanceId": "p:0a205n",
	  "taskName": "ReceiveTask2",
	  "data": "
	{\"BooleanCorrelationField\":\"true\",\"TextCorrelationField\":\"abc\
	",\"DateCorrelationField\":\"2022-12-
	15\",\"DecimalCorrelationField\":\"11.5\",\"IntegerCorrelationField\"
	:\"16783\",\"TimeCorrelationField\":\"13:30:00\",\"TimeoneCorrelation
	Field\":\"2022-12-
	18T05:20:00.000Z\",\"URICorrelationField\":\"http://ace=nightly-
	test\"}"
	}
	or
	{
	"correlationData": {
	"BooleanCorrelationField": true,
	"TextCorrelationField": "abc" },
	"taskName": "ReceiveTask2",
	"data": "
	{\"BooleanCorrelationField\":\"true\",\"TextCorrelationField\":\"abc\
	",\"DateCorrelationField\":\"2022-12-
	15\",\"DecimalCorrelationField\":\"11.5\",\"IntegerCorrelationField\"
	:\"16783\",\"TimeCorrelationField\":\"13:30:00\",\"TimeoneCorrelation
	Field\":\"2022-12-
	18T05:20:00.000Z\",\"URICorrelationField\":\"http://ace=nightly-
	test\"}"
	}

Response	Not applicable

Working with Work Item

Work items are the individual pieces of work within the system. A work item results from the process flow reaching a user task in the process; a work item is created (by private API calls that are restricted for internal use) and is sent to the resources specified as the participants of the user task.

Work Item States

A BPM work item is always in one of a small number of defined states.

Some of the operations used to manipulate work items cause an item to change its state, as defined in Work Item State Transitions. The following table shows the states that exist in the BPM runtime.

Work Item State	Meaning
(WMstateCreated)	This is a private state, accessed only by services that are restricted for internal use.
WMstateOffered	This is the initial state of every work item that is made available on the BPM runtime. The work item is offered to those resources who correspond to the organizational entities specified at design time, and is placed in their work list.
WMstateAllocated	The work item is assigned to a particular resource to be worked on. A work item may be in the Allocated state more than once in its life cycle, since it can be reallocated to different resources.
WMstateOpened	A work item is Opened when a resource (a user) begins work on it, either by selecting it from a work list or by having it automatically allocated and opened.
WMstateSuspended	A work item is suspended if its parent process instance is

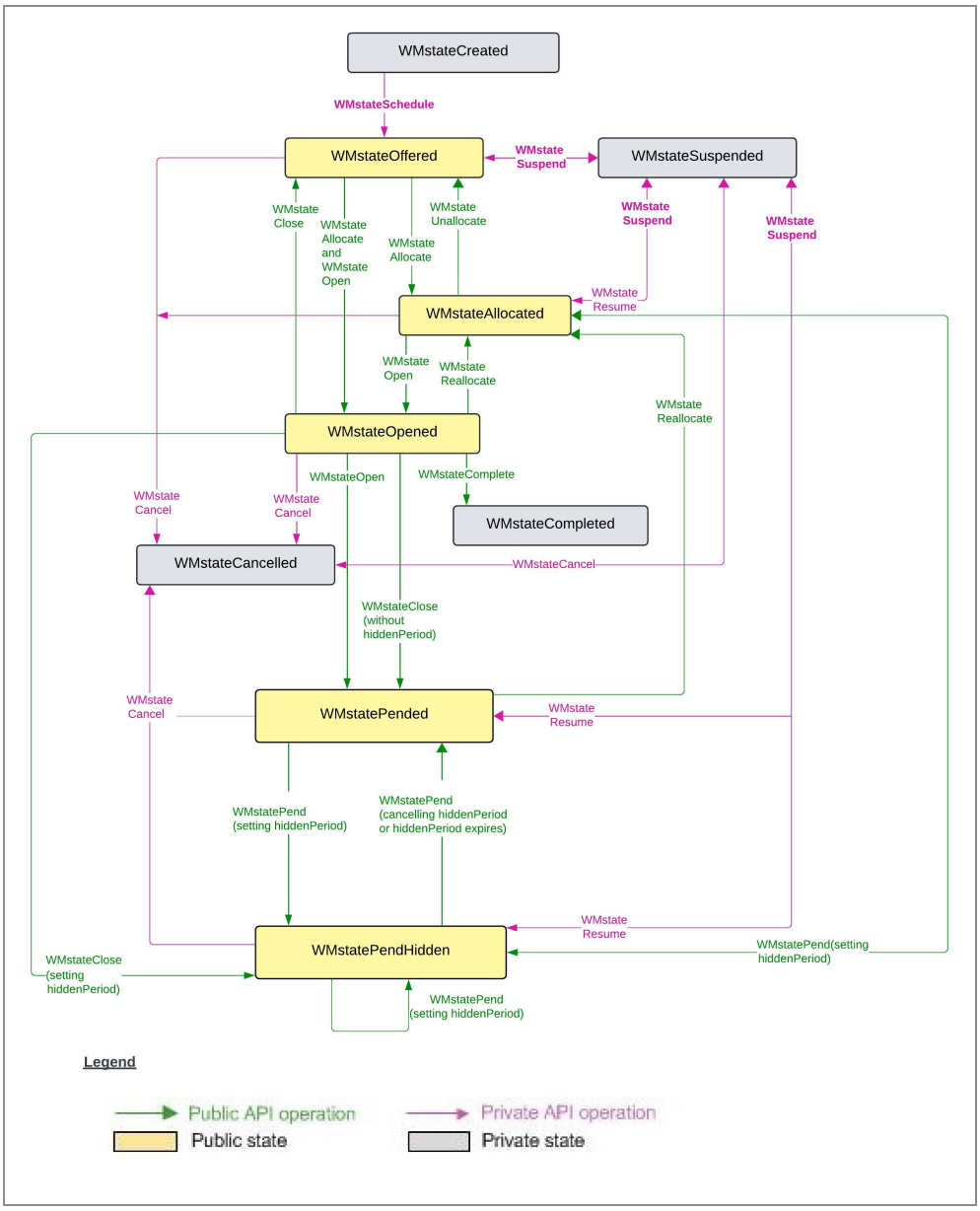
Work Item State	Meaning
	suspended.
WMstateCancelled	An open work item is cancelled when its parent process instance is cancelled. Work items that are not open are deleted when their parent process instance is cancelled. A cancelled work item can be closed by calling with no associated data. No other operations may affect it.
WMstatePended	The work item is assigned to a particular resource, who has done some work on it, but not yet completed that work. The work item's form has been opened and closed (not cancelled), and data fields may have been modified and saved.) **Note**: Pended and Allocated are similar states. The difference is that Pended means the work item *has* already been worked on. Allocated means that it has *not* already been worked on. You cannot skip a Pended work item.
WMstatePendHidden	If a work item is Pended with a hiddenPeriod specified, it cannot be accessed for the time defined in that hiddenPeriod. When the hiddenPeriod expires, the work item is returned to the state it was in before it was hidden. (An Allocated work item that was hidden returns to its Allocated state. A Pended work item that was hidden returns to its Pended state.)
WMstateCompleted	The final state of a work item in the BPM runtime, when work on an Opened work item has been completed. It is no longer on the system and cannot be affected by any API operations.

Work Item State Transitions

Work items go through transitions from state to state.

The following diagram and table show what transitions a work item goes through between states for PUT operation that is used to bring about each of the possible changes.

State Transitions



Start State	End State	Transition State	Description
WMstateOffered	WMstateAllocated	WMstateAllocate	The offered work item is **allocated** to the specified organization model entity.

Start State	End State	Transition State	Description
	WMstateOpened	WMstateAllocate and WMstateOpen	The offered work item is **allocated** to the specified organization model entity and immediately **opened**.
WMstateAllocated	WMstateOffered	WMstateUnallocate	The allocated object is returned to its original **offered** state.
	WMstateOpened	WMstateOpen	The allocated work item is **opened**.
	WMstatePendHidden	WMstatePend	The allocated work item is put into the **pendHidden** state for the duration of the specified *hiddenPeriod*. When the *hiddenPeriod* timer expires, the work item is returned to its original **allocated** state.
WMstateOpened	WMstateOffered	WMstateClose	The open work item (which must

Start State	End State	Transition State	Description
			contain no data changes) is closed and returned to its **offered** state.
	WMstateAllocated	WMstateReallocate	The open work item is reallocated to the specified organization model entity. It will be in the **allocated** state.
	WMstateCompleted	WMstateComplete	The opened work item is **complete**.
	WMstatePended	WMstateClose	The open work item is closed and any new data copied. It is then put into the **pended** state.
	WMstatePendHidden	WMstatePend	The open work item is closed and any new data copied. It is then put into the **pendHidden** state. When the *hiddenPeriod* timer expires, the work item is

Start State	End State	Transition State	Description
			transitioned to the **pended** state.
WMstatePended	WMstateAllocated	WMstateReallocate	The pended work item is reallocated to another organization model entity and put into the **allocated** state.
	WMstateOpened	WMstateOpen	The pended work item is **opened**.
	WMstatePendHidden	WMstatePend	The pended work item is put into the **pendHidden** state for the duration of the specified *hiddenPeriod*. When the *hiddenPeriod* timer expires, the work item is returned to its original **pended** state.
WMstatePendHidden	WMstatePended or WMstateAllocated	WMstatePend	A work item that was hidden using pendWorkItem is returned to the state it was in before it was

Start State	End State	Transition State	Description
			hidden - **pended** or **allocated**. A work item that was hidden is transitioned to the **pended** state.
	WMstatePendHidden	WMstatePend	The duration for which the work item will remain in the **pendHidden** state is reset to the specified *hiddenPeriod*.

Note: A work item cannot be accessed while it is in the **PendHidden** state.

Applications

Custom user interface applications are browser-based applications that consist entirely of static resources (such as HTML, CSS, JavaScript, XML, and JSON), which are served to the browser that hosts the application. Applications do not include servlets or client-side executables, such as applets or .NET libraries.

How to Access Application Development

You can access Application Development in a couple of different ways.

You can use the following URL:

```
protocol://host:port/apps/appdev/index.html
```

where:

- `protocol` is the communications protocol being used by Application Development, which is either http or https. This was determined at installation.
- `host` is the DNS name or IP address of the server hosting the BPM runtime.
- `port` is the port being used by the TIBCO BPM Enterprise server. The default value is 80.

Or you can use the steps below:

Procedure

1. Enter the following URL in your browser:

   ```
   protocol://host:port/apps/login
   ```

   where:

   - *protocol* is the communications protocol being used, either `http` or `https`. This

was specified at installation.

- *host* is the DNS name or IP address of the server hosting the TIBCO BPM Enterprise runtime.
- *port* is the port being used. The default value is 80.

2. Log in with a valid TIBCO BPM Enterprise username and password.
3. Click .
4. Click **App Dev**.

Application Lifecycle

An application can be created, edited, tested, and published. It can be re-edited and published again. Different users can do this, so you can have multiple versions of an application tailored to different users' requirements. The latest published application version is provided to the user.

You can do the following as part of the lifecycle of an application:

- Upload an application into Application Development (**New Upload**).

  Note: The application to be uploaded must have a `.zip` extension only.

- Create a blank application using **Create blank app**.
- Launch applications either before or after publishing them. This allows you to test your application. If you still need to make changes after testing, edit the application again, re-launch, and then publish (**Launch** either **Published** or **Latest**).
- Publish the application (so your changes can be seen by others) (**Publish**). Typically, you only do this once you are satisfied with the changes you have made and have tested them. Any user can now use the published application - or edit the published application and use their version locally.
- Show **Details** of your application. You can also see the version of the application you are using, view other versions available, revert to a version, or delete a version.
- **Download** an application. When you have downloaded the application, it will have the suffix `.zip`. You can extract the contents of the zip file and edit it locally.

- **Delete** the application you no longer require with the **Delete** button.
- **Browse** an application. You can manage it here, add and delete folders and files, and upload new files. You can also edit files here.
- **Clone** an application. You can clone all data from an application.

REST API

The TIBCO BPM Enterprise REST API provides RESTful interfaces.

Using the REST API, a client application can invoke BPM services using simple HTTP methods and intuitive URIs that identify BPM resources and the operations to be performed on them. Documentation for the REST API is provided in the *API Explorer*. Access the API Explorer as follows:

Procedure

1. Enter the following URL in your browser:

   ```
   protocol://host:port/apps/login
   ```

   where:

   - *protocol* is the communications protocol being used, either `http` or `https`. This was specified at installation.
   - *host* is the DNS name or IP address of the server hosting the TIBCO BPM Enterprise runtime.
   - *port* is the port being used. The default value is 80.

2. Log in with a valid TIBCO BPM Enterprise username and password.
3. Click .
4. Click **API Explorer**.

   Note: To access the REST API, you need to be logged in.

   Note: For additional details of each API, refer to the specifications of that API.

   For more details, refer to the authentication section below.

Authentication

An authenticated user is required to access TIBCO BPM Enterprise. Users must be registered with the TIBCO BPM Enterprise Directory Engine via the Organization Browser.

TIBCO BPM Enterprise supports the following types of authentication:

- Basic Authentication- The credentials used for authentication are obtained from the HTTP request in the form of a user name and password. The user name and password are authenticated against an LDAP.
- SAML Web Profile - If your TIBCO BPM Enterprise application is configured to use SAML Web Profile for authentication, users of your application can log in using a user name and password issued by an Identity Provider (IdP) that supports SAML Web Profile.
- OpenID Connect - If your TIBCO BPM Enterprise application is configured to use OpenID Connect, the users can log in with a user name and password issued by an Identity Provider (IdP) that supports OpenID Connect.

Authentication Process

TIBCO BPM Enterprise contains a login module for each of the available types of authentication; basic, SAML Web Profile, and OpenID Connect. When a TIBCO BPM Enterprise HTTP endpoint is accessed, the appropriate login module handles the user authentication by performing the following steps:

1. The system checks for a current user session, and whether or not it has expired. If a current user is in session, the HTTP request is processed.
2. If there is no current user session, a check is made to determine if TIBCO BPM Enterprise is configured for basic authentication. Basic authentication is HTTP basic authentication. In HTTP basic authentication, the principal's credentials are passed in the HTTP Authorization request header.

   The basic authentication login module extracts the principal from the HTTP authorize header (if it is available) and searches for the user in TIBCO BPM Enterprise system. If the user exists in TIBCO BPM Enterprise, the system returns details of the user, including the primary LDAP to be used for authentication purposes.

   Basic authentication is configured using an HTTP Client Shared Resource defined in TIBCO BPM Enterprise Administrator.

3. If basic authentication is not used or fails, the system checks if TIBCO BPM Enterprise is configured for Single Sign-On (SSO) authentication (SAML Web Profile or OpenID Connect). SSO authentication must be configured if a basic authentication is not configured. Also, only one of the SSO authentication type configurations is supported across all in-bound TIBCO BPM Enterprise REST APIs at a given time (although, both types can be configured, only one can be enabled at a time).

   Depending on which SSO authentication type is configured, control is handed over to the appropriate login module (SAML Web Profile or OpenID Connect), which uses the appropriate shared resource configuration defined in TIBCO BPM Enterprise Administrator.

   After SSO authentication is completed, an authorization check is performed to ensure that the user exists in TIBCO BPM Enterprise. This is done by looking up the user in Directory Engine. If this is successful, the user is considered as authentic and an HTTP session is created.

SAML Web Profile Authentication

If your TIBCO BPM Enterprise application is configured to use SAML Web Profile for authentication, users can log in with a username and password issued by an IdP that supports SAML Web Profile. TIBCO BPM Enterprise supports Google and simpleSAMLphp SAML IdP.

> **Note:** Ensure that the resource registered with your IdP is added to the LDAP.

Perform the following procedure to ensure that SAML authentication works with your registered users:

1. Set up your preferred SAML Idp to download to your local machine. For more information, visit the website of your IdP provider.
2. Configure your SAML Idp. For more information about configuring a SAML shared resource, see SAML Authentication Shared Resources.
3. Ensure that the user whose login credentials are registered with the Idp is also added to the LDAP Container. For more information, see the Configure the LDAP Directory Server topic in the *TIBCO BPM Enterprise Installation Guide*.

The following steps describe the basic flow when a user attempts to log in to a TIBCO BPM Enterprise application, which is configured to use SAML Web Profile, using their IdP credentials. In this scenario, the user is not already logged in to TIBCO BPM Enterprise.

1. The user starts a TIBCO BPM Enterprise application that is using SAML Web Profile authentication.
2. The application tries to access the TIBCO BPM Enterprise server, but the login module determines that the user is not authenticated and that authentication is provided by SAML Web Profile.
3. The application redirects the login request to the IdP.
4. The IdP displays a login screen (for example, Google's login screen), requesting the user's IdP-issued credentials.
5. The user enters their IdP-issued credentials.
6. Upon receiving the user validation from the IdP, the application redirects the request to the TIBCO BPM Enterprise server to authenticate the user before logging the user in to the application.

A cookie is also created when the user is validated by the TIBCO BPM Enterprise server. The cookie is used to establish the session that is used by all subsequent calls to the TIBCO BPM Enterprise server.

The following steps describe the events that occur when an IdP-authenticated user logs out of a TIBCO BPM Enterprise application:

- The user is redirected to the login page for the application. When the request is redirected to `<domain>/apps/login/index.html`, the login page checks for an existing authenticated session. If there is no authenticated session, it forwards the request to the SAML IdP provider login page (if the user is not authenticated with the IdP).
- The cookie that was created upon login is removed.

OpenID Connect Authentication

If your TIBCO BPM Enterprise application is configured to use OpenID Connect, users of your application can log in using a username and password issued by an Identity Provider (IdP) that supports OpenID Connect.

The following describes the basic flow when someone attempts to log in to an TIBCO BPM Enterprise application, which is configured to use OpenID Connect, using their IdP credentials (this assumes the user is not already logged in to TIBCO BPM Enterprise):

1. A user starts a TIBCO BPM Enterprise application that is using OpenID Connect authentication.
2. The application tries to access the TIBCO BPM Enterprise server, but the login module determines that the user is not authenticated, and that authentication is being provided by OpenID Connect.
3. The application redirects the login request to the IdP.
4. The IdP displays their login screen, requesting the user's IdP-issued credentials.
5. The user enters IdP-issued credentials.
6. After validating the user, the IdP returns an ID Token — in the form of a JSON Web Token (JWT) — to indicate a successful authentication.

   > **Note:** Using the OpenID **Access Token** is not currently supported in but the login module determines that the user is not authenticated and that authentication is being provided. The OpenID **ID Token** is used to identify the user.

   The response from the IdP also includes the *claims* specified in the **Auth Scope** field of the OpenID Authentication shared resource.

   The IdP sends the ID Token and claims information to the "Redirect URI" that is specified in the OpenID Connect shared resource.
7. Upon receiving the ID Token from the IdP, the application redirects the request back to the TIBCO BPM Enterprise server to confirm that the user is a valid TIBCO BPM Enterprise user before logging the user into the application.

A cookie is also created when the user is validated by the TIBCO BPM Enterprise server. The cookie includes the ID Token, which is used to establish the session that is used by all other subsequent calls to the TIBCO BPM Enterprise server.

When an IdP-authenticated user logs out of the TIBCO BPM Enterprise application:

- The browser sends the value in the **Logout path** property to the TIBCO BPM Enterprise server. (When a user logs out, the user does not log out of the IDP but only invalidates the client session.)

- The cookie that was created upon login is removed.

Note: At any point, only a single SSO related shared resource can be enabled, that is, either SAML or OpenID.

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

The documentation for this product is available on the TIBCO® BPM Enterprise 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, Business Studio, TIBCO Business Studio, and Spotfire 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 © 2015-2024. Cloud Software Group, Inc. All Rights Reserved.