TIBCO® Workspace Components Developer Guide

Software Release 4.2
August 2017

Two-Second Advantage®

Important Information

Contents

TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, visit:

https://docs.tibco.com

Product-Specific Documentation

Documentation for TIBCO products is not bundled with the software. Instead, it is available on the TIBCO Documentation site. To directly access documentation for this product, double-click the following file:

`TIBCO_HOME/release_notes/TIB_amx-bpm_version_docinfo.html`

where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is `C:\tibco`. On UNIX systems, the default *TIBCO_HOME* is `/opt/tibco`.

The following documents for this product can be found on the TIBCO Documentation site:

- TIBCO ActiveMatrix BPM SOA Concepts
- TIBCO ActiveMatrix BPM Concepts
- TIBCO ActiveMatrix BPM Developer's Guide
- TIBCO ActiveMatrix BPM Web Client Developer's Guide
- TIBCO ActiveMatrix BPM Tutorials
- TIBCO ActiveMatrix BPM Business Data Services Developer Guide
- TIBCO ActiveMatrix BPM Case Data User Guide
- TIBCO ActiveMatrix BPM Event Collector Schema Reference
- TIBCO ActiveMatrix BPM - Integration with Content Management Systems
- TIBCO ActiveMatrix BPM SOA Composite Development
- TIBCO ActiveMatrix BPM Java Component Development
- TIBCO ActiveMatrix BPM Mediation Component Development
- TIBCO ActiveMatrix BPM Mediation API Reference
- TIBCO ActiveMatrix BPM WebApp Component Development
- TIBCO ActiveMatrix BPM Administration
- TIBCO ActiveMatrix BPM Performance Tuning Guide
- TIBCO ActiveMatrix BPM SOA Administration
- TIBCO ActiveMatrix BPM SOA Administration Tutorials
- TIBCO ActiveMatrix BPM SOA Development Tutorials
- TIBCO ActiveMatrix BPM Client Application Management Guide
- TIBCO ActiveMatrix BPM Client Application Developer's Guide
- TIBCO Openspace User's Guide
- TIBCO Openspace Customization Guide
- TIBCO ActiveMatrix BPM Organization Browser User's Guide (Openspace)
- TIBCO ActiveMatrix BPM Organization Browser User's Guide (Workspace)

- TIBCO ActiveMatrix BPM Spotfire Visualizations
- TIBCO Workspace User's Guide
- TIBCO Workspace Configuration and Customization
- TIBCO Workspace Components Developer Guide
- TIBCO ActiveMatrix BPM Troubleshooting Guide
- TIBCO ActiveMatrix BPM Deployment
- TIBCO ActiveMatrix BPM Hawk Plug-in User's Guide
- TIBCO ActiveMatrix BPM Installation: Developer Server
- TIBCO ActiveMatrix BPM Installation and Configuration
- TIBCO ActiveMatrix BPM Log Viewer
- TIBCO ActiveMatrix BPM Single Sign-On
- Using TIBCO JasperReports for ActiveMatrix BPM

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

https://community.tibco.com

Introduction

The TIBCO® Workspace components (simply called the “Workspace components” or "WCC components"(1) throughout the documentation) provide a set of data-aware user interface components that may be used to quickly develop full-screen web applications, embed functionality in web pages, or serve as portlets in a portal application. They allow the developer to quickly incorporate functionality into BPM-aware AJAX applications without requiring a working knowledge of the API's that are needed to access and manage the data.

The Workspace components are created with TIBCO General Interface Builder and are composed of standard General Interface components that have been integrated and enhanced to provide the functionality and user interfaces required of a typical BPM client application. At design-time, the components are available as drag and drop tools in the Component Libraries of the General Interface Builder development environment.

The following are examples of the functionality you can add to your web application with the Workspace components:

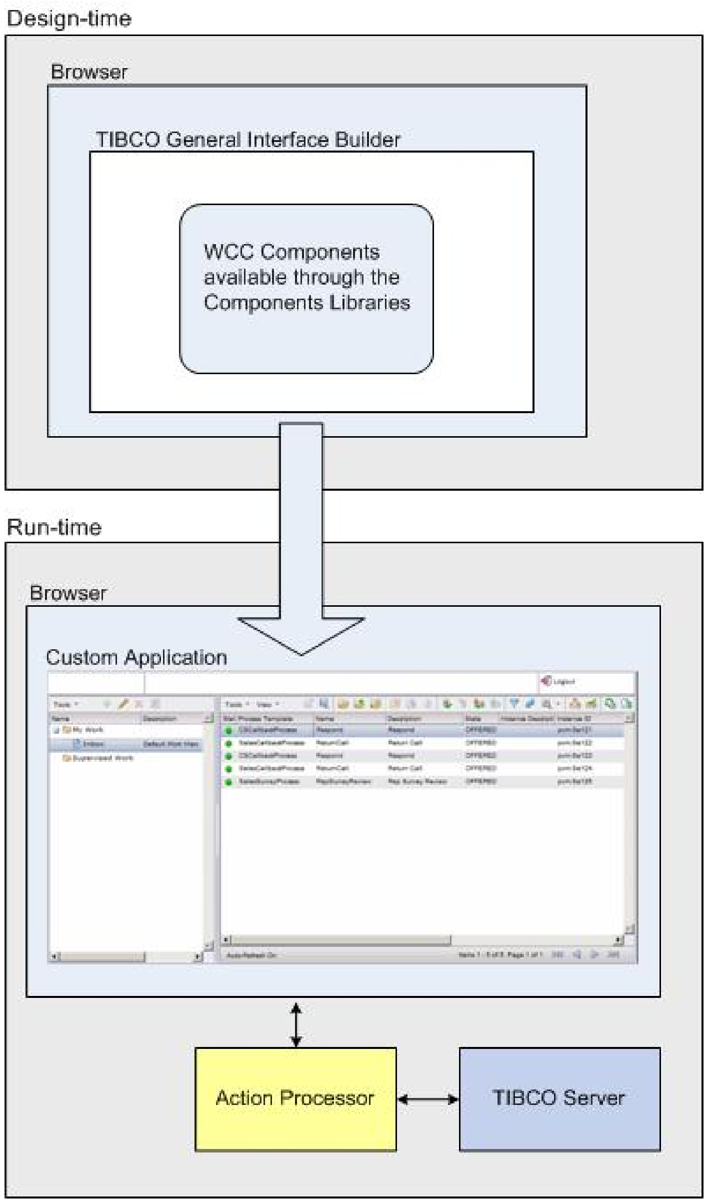
- Log into, and out of, the server.
- Display lists of process templates, process instances, work items, and events.
- Filter or sort lists of process instances, work items, and events.
- Start business services.
- Process (open, save, submit, etc.) work items.
- Display the Organization Browser, which is used to map resources to groups and positions in the organization model.

A complete list of the functions available through the Workspace components can be found in Component Reference.

For information about using the functions available through the Workspace components, refer to the *TIBCO Workspace User’s Guide.*

The following illustration shows the major items/products that are used in a custom WCC application environment.

[1] The Workspace components are often referred to as “WCC” components, which stands for “Workspace Client Components”. You will see the term WCC used throughout the documentation. "Workspace components" and "WCC components" are synonymous. And a "custom WCC application" is a custom application created with WCC components.


Design-time
Browser
TIBCO General Interface Builder
WCC Components available through the Components Libraries
Run-time
Browser
Custom Application
Action Processor
TIBCO Server

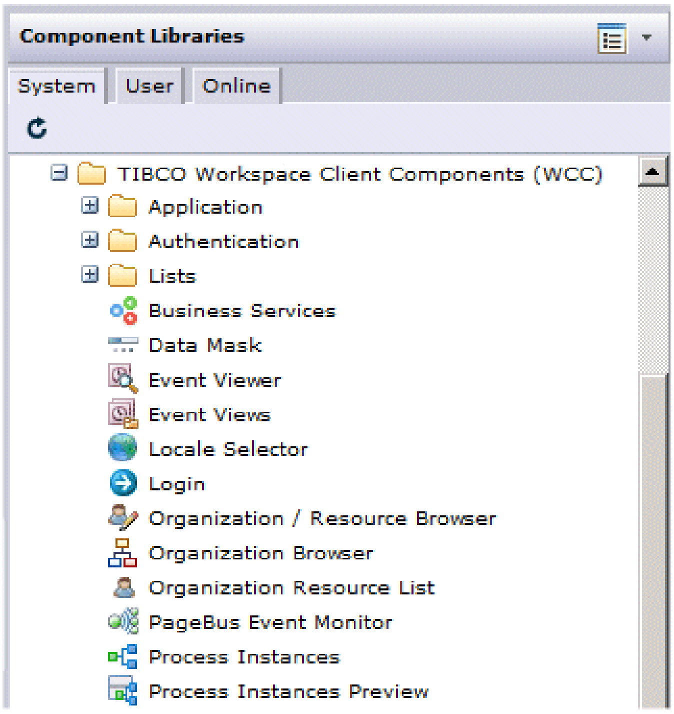
TIBCO General Interface

TIBCO General Interface consists of two components: General Interface Builder and General Interface Framework.

- **TIBCO General Interface Builder** - This is a browser-based, visual development environment used to develop AJAX applications. AJAX (asynchronous communications, JavaScript, and XML) applications are run in a standard browser, but look and feel like a desktop-run application.

  General Interface Builder includes a Components Libraries Palette that contains many standard components used to add functionality to your application, such as menus, taskbars, text boxes, etc.

  After Workspace components are installed and added to your project, they are also available in the General Interface Builder Components Libraries. The Workspace components provide the same functionality that is available in the Workspace application.



Information about using and configuring Workspace components in General Interface Builder is provided in the remainder of this guide.

- **TIBCO General Interface Framework** - This is a distributable runtime framework for running browser-based applications developed with General Interface Builder.

  Note that this framework is deployed as part of Workspace — it does *not* need to be deployed to client machines to run a WCC application. The only software need to run the application locally is a browser.

This release of Workspace was built with TIBCO General Interface version 3.9.2.

TIBCO General Interface is installed as part of TIBCO Business Studio, which is used to create and deploy business processes. Therefore, before you can begin developing custom WCC applications, you must install TIBCO Business Studio, then start TIBCO General Interface Builder — for more information about this process, see the How to Create a BPM Desktop Using Components Tutorial.

Browsers

AJAX applications developed using TIBCO General Interface Builder and Workspace components can be run in most common Internet browsers. For a complete list of the supported browsers, see the *TIBCO ActiveMatrix BPM Installation* guide.

No other software (applets, plug-ins, or Active X controls) is required to run applications developed with Workspace components.

Note, however, that cookies must be enabled in the browser from which the application is being run. Cookies are used to perform session tracking.

Also, for information about the browsers that can be used to *develop* applications using TIBCO General Interface Builder, see the TIBCO General Interface documentation.

Custom Application

AJAX applications developed using the General Interface Builder and Workspace components can be full-screen applications, portlets in a portal application, or a specific piece of functionality embedded in a web page.

The size of the component interface displayed to the user is controlled by the container in which the component is placed.

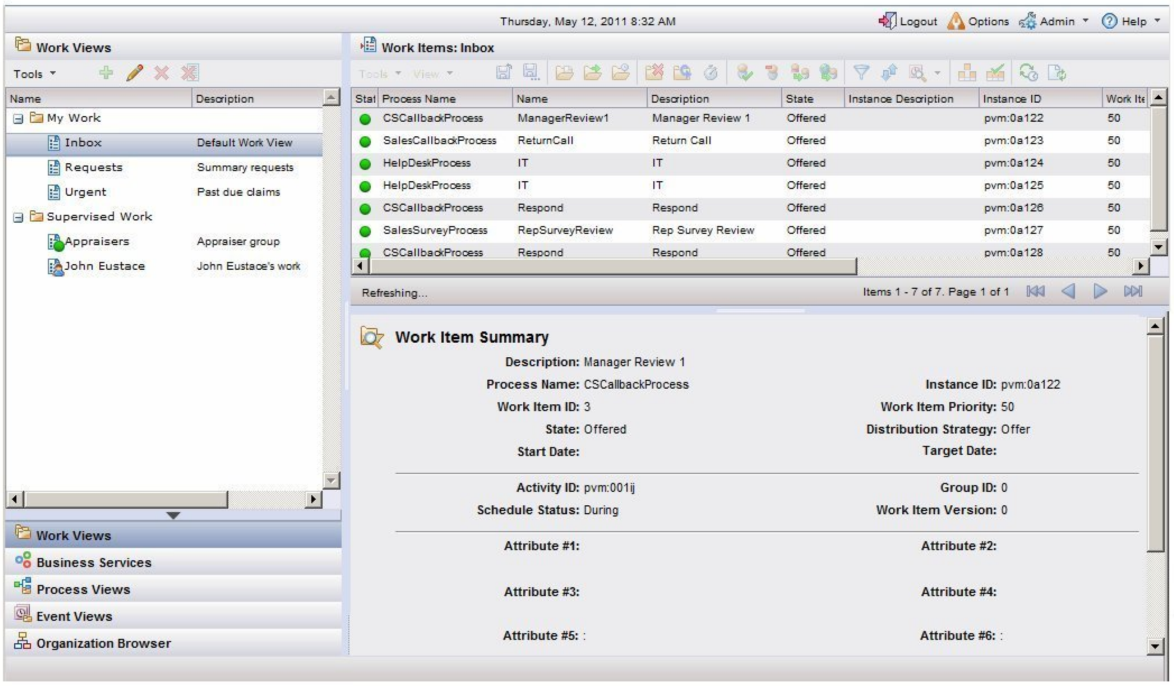
After a custom application is created using General Interface Builder and WCC components, it can then be deployed to a node. Users can access the custom application via a browser.

For details about creating a custom application using WCC components, see Building Custom Applications.

Workspace

TIBCO produces a "Workspace" application that incorporates all functions that are available using WCC components. This web application allows the user to start business services, view work items, process work items, etc.

The illustration below shows the Workspace application user interface.



Name	Description
My Work	
Inbox	Default Work View
Requests	Summary requests
Urgent	Past due claims
Supervised Work	
Appraisers	Appraiser group
John Eustace	John Eustace's work

Stat	Process Name	Name	Description	State	Instance Description	Instance ID	Work Ite
●	CSCallbackProcess	ManagerReview1	Manager Review 1	Offered		pvm:0a122	50
●	SalesCallbackProcess	ReturnCall	Return Call	Offered		pvm:0a123	50
●	HelpDeskProcess	IT	IT	Offered		pvm:0a124	50
●	HelpDeskProcess	IT	IT	Offered		pvm:0a125	50
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a126	50
●	SalesSurveyProcess	RepSurveyReview	Rep Survey Review	Offered		pvm:0a127	50
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a128	50

Refreshing... Items 1 - 7 of 7. Page 1 of 1

Work Item Summary

Description: Manager Review 1
Process Name: CSCallbackProcess **Instance ID:** pvm:0a122
Work Item ID: 3 **Work Item Priority:** 50
State: Offered **Distribution Strategy:** Offer
Start Date: **Target Date:**

Activity ID: pvm:001ij **Group ID:** 0
Schedule Status: During **Work Item Version:** 0

Attribute #1: **Attribute #2:**
Attribute #3: **Attribute #4:**
Attribute #5: : **Attribute #6:** :

Details about the functions available in this application (which are also the functions available through the Workspace components) can be found in the *TIBCO Workspace User's Guide*.

The Workspace application is also installed as part of TIBCO Business Studio. This allows you to open the Workspace application and modify it with General Interface Builder, if desired.

Workspace Action Processor

The Workspace Action Processor services action requests made by applications developed using Workspace components.

Each 'action' requested by a Workspace component can contain one or more of the following 'request' types:

- Request for data (e.g., a list of work items)

- Request an action be executed (e.g., start an instance of a process template)

The components send their action details to the Action Processor via a Form POST over HTTP — the action details are sent as XML on an HTML <input> tag.

The Action Processor also accepts actions via an HTTP GET. When doing a GET, the action XML needs to be sent as a URL parameter named 'action'.

Upon receiving an action request, the Action Processor breaks the action into individual requests and processes each one. The resulting data from each request is then consolidated in XML under one root element (<ap:ActionResult>). The Action Processor then returns this XML data to the components over HTTP (or in the case of a form request, it redirects the HTML — no XML is returned).

The Action Processor runs on the same machine as, and is installed as part of, the TIBCO ActiveMatrix BPM Node.

TIBCO ActiveMatrix BPM Node

This consists of the backend components that process requests from the Workspace.

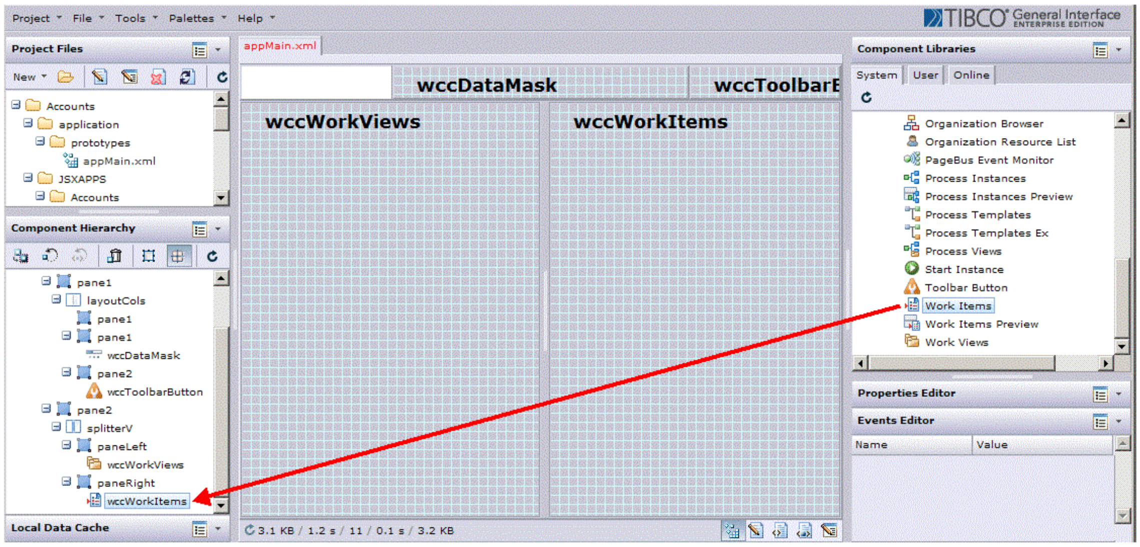
It performs functions such as the following:

- Executes processes that have been deployed from TIBCO Business Studio.
- Is responsible for delivering work items to the appropriate users.
- Maintains a model of the organizational structure of the enterprise, which it augments with the necessary BPM-related information required to manage work for users.
- Displays work items to users so that they can work on and complete them, and for managing the dialogue associated with performing the work required.

For more details, see the *TIBCO ActiveMatrix BPM Concepts* guide.

Building Custom Applications

Building a custom application with Workspace components is as easy as dragging and dropping the appropriate “WCC” (workspace client component) components into containers in General Interface Builder, then subscribing to the appropriate events using the Events Editor.



Location of Custom Applications

When TIBCO General Interface Builder is started to create a custom WCC application, you are asked for a General Interface *workspace* directory. You must specify that the workspace directory is the same directory in which General Interface Builder is installed, which is the directory in which the `GI_Builder.html` file is located.

This defaults to:

`*StudioHome*\wcc\*version*\GI_Builder.html`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.

This causes your custom application to be created in the following location:

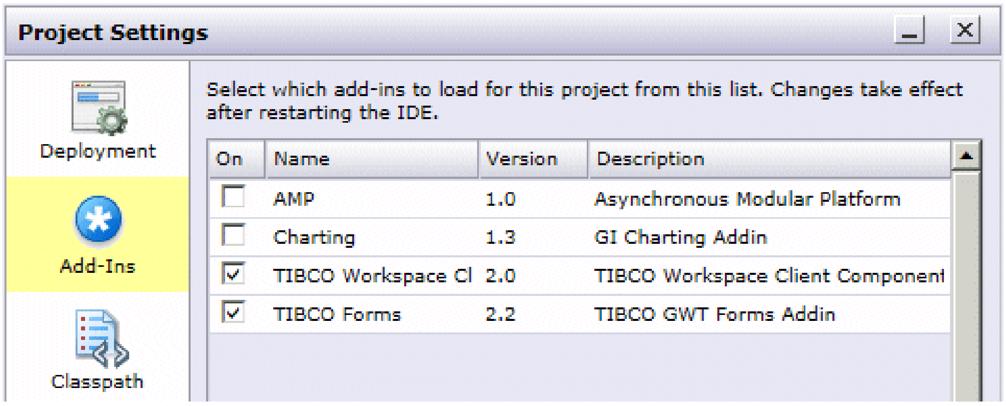
`*StudioHome*\wcc\*version*\JSXAPPS\*YourCustomAppName*`

For more information, see the tutorial for creating a custom WCC application in Tutorial Procedure.

WCC Add-in

Before you can add WCC components to your custom application, you must include the “WCC Add-in” in your TIBCO General Interface project. This add-in gives you access to the WCC components in the General Interface Builder Component Libraries when you are developing your custom application.

To include the WCC Add-in in your project, ensure that the **TIBCO Workspace Client Components (WCC)** box is checked on the Project Settings / Add-ins dialog:


Project Settings

Deployment
Add-Ins
Classpath

Select which add-ins to load for this project from this list. Changes take effect after restarting the IDE.

On	Name	Version	Description
☐	AMP	1.0	Asynchronous Modular Platform
☐	Charting	1.3	GI Charting Addin
☑	TIBCO Workspace Cl	2.0	TIBCO Workspace Client Component
☑	TIBCO Forms	2.2	TIBCO GWT Forms Addin

For more information about including the WCC Add-in and creating a custom WCC application using General Interface Builder, see the How to Create a BPM Desktop Using Components Tutorial.

Application Template

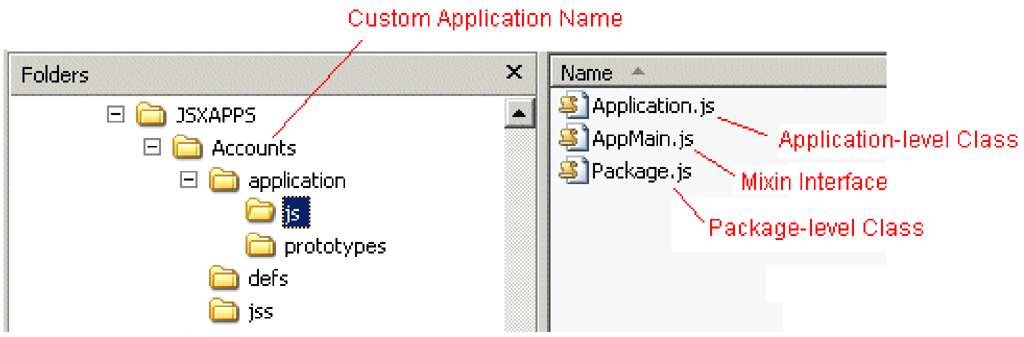
When a custom application containing WCC components is created using TIBCO General Interface Builder, a template is used to create the directory structure and classes necessary for your custom application.

For example, if you create a custom application called “Accounts”, the structure shown in the illustration below is created under:

StudioHome\wcc*version*\JSXAPPS\Accounts

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.



The classes that are created for your custom application are described below:

Application-level class — Application.js

This is the main application-level class for your project. The `Application.js` class:

- Provides entry points to add custom code into your custom WCC application. These are in the form of override methods, such as init, postLoadInit, postLogin, etc.

  For more information about the available override methods, see Override Methods.
- Extends the “base” Application class (`com.tibco.wcc.components.Application`), which provides access to various public methods that allow you to do things like get the application namespace (**getNameSpace**), get the `jsx3.app.Server` for the application (**getServer**), etc.

Extending the `com.tibco.wcc.components.Application` class also provides access to public methods that perform functions normally performed by components, such as opening a work item, suspending a process instance, etc.

For information about these methods, see Application Class Methods.

- Implements the `com.tibco.wcc.`*`WCCProjectName`*`.AppMain` class, a mixin interface, which is used to separate custom code from the main Application class.

Mixin interface — AppMain.js

The mixin interface (`AppMain.js`) is implemented by the main Application class for your custom application. It contains samples of how custom methods can be added. Plus, it provides a container for other custom code you want to add.

For details about the method examples in the mixin interface, see Mixin Interface Methods.

Package-level class — Package.js

This class is provided to include package-level custom code, if desired.

Application Configuration

Before deploying your custom WCC application, configuration is accomplished by making changes to configuration files in your local development environment.

For information, see the "Configuring a Deployed Application" topic in the *TIBCO Workspace Configuration and Customization* guide.

After your custom WCC application is deployed, configuration is accomplished using the Configuration Administrator. For information, see the "Configuring a Deployed Application" topic in the *TIBCO Workspace Configuration and Customization* guide.

Running a Custom Application

While you are creating, customizing, and configuring your custom WCC application, you will be testing it by running it from your local file system. This can be done simply by executing the launch fragment created by General Interface Builder.

This launch fragment is located as follows:

`StudioHome``\wcc\`*`version`*`\`*`WCCProjectName`*`.html`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.
- *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".

For example, if you create a custom application called “Accounts”, an `Accounts.html` file is created in your workspace directory. Executing this file starts the custom application.

Note, however, before you can execute the launch fragment, you must modify the custom application’s `config.xml` file, to provide a URL to the TIBCO ActiveMatrix node/ Action Processor. This is done as follows:

Procedure

1. Open the custom application’s `config.xml` file, which is located as follows:

   `StudioHome``\workspace\`*`version`*`\JSXAPPS\`*`WCCProjectName`*`\config.xml`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.
- *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".

2. Locate the **ActionProcessors** record in the `config.xml` file.
3. Set the **baseUrl** attribute to the URL of the Action Processor. The string in the **baseUrl** attribute must be in the form:

   ```
   http://Host:Port/bpm/actionprocessor/actionprocessor.servlet
   ```

   where:

   - *Host* is the name or IP address of the machine hosting the BPM runtime.
   - *Port* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

     For example:

```
<record jsxid="ActionProcessors" type="workspace">
    <ActionProcessor
       weighting="100"
       baseUrl="http://Austin:8080/bpm/actionprocessor/actionprocessor.servlet">
    </ActionProcessor>
</record>
```

Note - The **weighting** attribute is not used at this time.

4. Save and close the `config.xml` file.
5. You can now run your application by executing the launch fragment as described at the beginning of this topic.

For information about launching a deployed application, see the *Introduction* chapter in the *TIBCO Workspace Configuration and Customization guide.*

Cross-Domain Scripting

Cross-domain scripting is a security vulnerability of web applications. If you trigger cross-domain scripting, and your browser doesn't allow it, the web application will not run (in the case of a WCC application, it will state that it is unable to establish a connection to the Action Processor).

Some browsers are more strict about enforcing cross-domain scripting than others; and newer versions of browsers tend to be more strict than older versions. Some browsers also provide methods to allow cross-domain scripting—see your browser's documentation for more information.

Cross-domain scripting affects accessing WCC applications in the following ways:

- **URL used to launch the application** - To prevent cross-domain scripting, it is best practice to ensure that the domain portion of the URL that is entered into the address line of the browser *exactly* matches the domain portion of the Action Processor URL specified in the application's `config.xml` file.

  The domain consists of the "`http://`*Host*`:`*Port*" part of the URL.

  The domain used to launch the application cannot differ in any way from the Action Processor's specified domain, otherwise cross-domain scripting may be triggered (depending on your browser). That is, you cannot use "http" in one and "https" in the other; you cannot use a host name in one and an IP address in the other; one host name cannot be unqualified and the other qualified; you cannot use "localhost" in one and "127.0.0.1" in the other.

  To determine if cross-domain scripting is being used, the browser simply compares the URL domains as strings.

Note that in a production environment, where the WCC application and Action Processor are deployed to the same node and HTTP connector, the normal practice is to specify an empty string for the Action Processor URL in the application's `config.xml` file. When this is done, the URL of the Action Processor is inferred from the URL used to launch the application. This avoids the issue of comparing URL strings. For more information, see the "Action Processor" topic in the *TIBCO Workspace Configuration and Customization* guide.

- **Running the application from the local file system** - Because of the security risk of cross-domain scripting, some browsers will not allow you to run a web application (including a WCC application) from the local file system.

  Note that you would typically only run a WCC application from the file system in a testing and development environment. In a production environment, it is expected that the application will be deployed to a Web server and run from there. For information about deploying, see the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Configuration and Customization* guide.

Launching a Custom WCC Application in an HTML Frame

To be able to launch a custom WCC application in an HTML frame (for example, an iframe in a portal), you must make some modifications to the launch fragment for the application.

Prior to launching the application in a frame, perform the procedure below.

Procedure

1. Open the launch fragment.
2. Locate the following: "NOTE: To allow display of this application under frames remove the following style and script elements".
3. Remove (or comment out) the <styleand> <script> elements immediately after the note. For example:

```
<!-- NOTE: To allow display of this application under frames remove the following
style and script elements
<style type="text/css">html{display:none;}</style>
<script language="javascript">
   if (self == top) {
        // Not in frame so show client app
        document.documentElement.style.display='block';
   } else {
        // In a frame so try to show client app outside of a frame
        top.location = self.location;
   }
</script> -->
```

4. Locate the following: "NOTE: To allow display of this application under frames remove the next script element".
5. Remove (or comment out) the <script> element immediately after the note. For example:

```
<!-- NOTE: To allow display of this application under frames remove the next
script element
<script language="javascript">
   if (self !== top) {
      // Still in a frame so clear body of app and close
      document.getElementsByTagName("body")[0].innerHTML = 'Not allowed in
frames.';
       window.open('close.html', '_self');
   }
</script> -->
```

6. Save and close the launch script.

Using a Load Balancer

If you have installed a BPM system using a distributed or highly available, fault tolerant configuration, you must configure the system to be able to communicate properly with your WCC application via the load balancer.

Information about configuring client communications through the load balancer can be found in the "Configure Client Communications Through the Load Balancer" topic in the *TIBCO ActiveMatrix BPM Installation and Configuration* guide.

To configure your custom WCC application to communicate via a load balancer:

Procedure

1. Perform the steps in Task A as they are described in the "Configure Client Communications Through the Load Balancer" topic.
2. For Task B in the "Configure Client Communications Through the Load Balancer" topic, you must configure your WCC application's launch fragment in your development environment, rather than from the ActiveMatrix BPM node as described in steps of Task B.
3. After configuring your application's launch fragment in the development environment, you must repackage your application in a `.war` file, then redeploy it to the node. This is described in the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Configuration and Customization* guide.

Deploying a Custom Application

When you have completed creating, customizing, configuring, and testing your custom WCC application locally, it can then be deployed to a node. This is accomplished using the provided WAR-file creation utility.

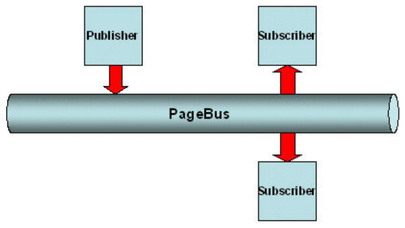
Using the WAR-file creation utility to deploy an application is described in the "Introduction" chapter in the *TIBCO Workspace Configuration and Customization* guide.

Component Events

Workspace components use events to publish actions and to pass business data through the communications channel. This is accomplished using the **TIBCO PageBus™**.

The PageBus is a pure JavaScript, publish-subscribe message delivery hub that uses string *topics* to identify business-level events, such as a user's selection of a work item from a work item list, or when a user re-allocates a work item.

Subscribers listen (or *subscribe*) to topics. Publishers send (or *publish*) messages on topics. When publishers publish messages on a topic, the messages are delivered to all of the topic's current subscribers.



When you add a Workspace component to your application, it is automatically configured to publish all of its public events to the PageBus. Then, for each component you've added, you must specify which events you want that component to subscribe to. This is accomplished using the Events Editor in TIBCO General Interface Builder — the Events Editor shows you the events available from all of the components that have been added to the application. (It also allows you to import event configurations from external applications, and subscribe to those.) For information about using the Events Editor, see Events Editor.

For more information about the TIBCO PageBus, see the *TIBCO PageBus™ Developer's Guide.*

Loading PageBus

WCC applications automatically load the TIBCO PageBus. However, it first looks to see if PageBus has already been loaded by another application on the system. If it is already loaded, the WCC application does not load it again.

Note, however, that WCC applications are tied to a specific version of PageBus. If the PageBus has already been loaded by another application, and it is not compatible with the WCC application, it may result in errors.

You can determine the version of PageBus required by the current WCC application by looking in the following file:

```
StudioHome\wcc\version\JSXAPPS\components\PageBus.js
```

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio

Publishing and Subscribing to Events

Communication between components in your application is accomplished by publishing, and subscribing to, events.

For example:

- **WCC component <-> WCC component** - When communicating between WCC components, the Events Editor is used to set up the connection between the components. There is no coding required. When you add a WCC component to your application, the events available from that component are automatically pushed to the PageBus. When you add another WCC component, the Events Editor shows you all of the events that are available to which that component can subscribe.

  For information about using the Events Editor, see Events Editor.
- **WCC component <-> WCC component in external application** - WCC components can also subscribe to events published by other WCC components that reside in an external application. This involves creating an *event definition file* (using the Application Publish Definer utility — see Events From External Applications) that defines the events published by the WCC components in the external application. The event definition file can then be *imported* (using the Events Editor) so that a WCC component can subscribe to the events from the external application.

  For more information, see Events From External Applications.
- **non-WCC component <-> WCC component** - When communicating between a non-WCC component and a WCC component, you will need to use the PageBus API to either publish to, or subscribe from, the PageBus, as follows:
  - **non-WCC component publishing an event** - Non-WCC components can publish events to the PageBus using the **PageBus.publish** method. WCC components can then subscribe to those events.

    For more information, see Non-WCC Components Publishing Events.
  - **non-WCC component subscribing to an event** - Non-WCC components can subscribe to events on the PageBus that have been published by WCC components. This is done using the **PageBus.subscribe** method.

    For more information, see Non-WCC Components Subscribing to Events.
- **non-WCC component <-> non-WCC component** - When communicating between non-WCC components, you must use the PageBus **publish** and **subscribe** methods to publish to, and subscribe from, the PageBus. For more information, refer to the *TIBCO PageBus Developer's Guide.*

Non-WCC Components Publishing Events

Non-WCC components can publish events to the PageBus using the **PageBus.publish** method. WCC components can then subscribe to those events.

The **PageBus.publish** method passes the following parameters:

Name	Type	Description
topic	string	Topic name on which to publish the message. This must not be a wildcard topic.
message	any	Message content. This data value *should* be serializable as JSON. If the event cache is used, this data value *must* be serializable as JSON.

For WCC components, the *topic* has the form:

```
com.tibco.wcc.appModelName.prototypeModelName.componentModelName.eventName
```

where:

- `com.tibco.wcc.appModelName` - The application model name (see: `app.getModelName()`)
- `prototypeModelName` - The model name given to the prototype that contains the component, `wccPrototype`, by default.

- `componentModelName` - The model name given to the component, `wccWorkItems`, for example.

The *message* parameter is a JavaScript object and has the form:

```
{
     jss  : "org.pagebus.msg.Message",
     jssv : "2.0.0",
     head : {topic : eventName},
     body : bodyObject
}
```

where:

- *bodyObject* is a JavaScript object that contains the data expected by the WCC component that will subscribe to that event.

  For information about the data in this object for each event, see Component PageBus Event Payloads.

For more details about using the **publish** method, see the *TIBCO PageBus™ Developer's Guide.*

Non-WCC Components Subscribing to Events

Non-WCC components can subscribe to events on the PageBus by using the **PageBus.subscribe** method. When a topic to which the non-WCC component has subscribed is published to the PageBus, the callback function specified in the **subscribe** method is invoked.

The **PageBus.subscribe** method passes the following parameters:

Name	Type	Description
topic	string	Topic name to which to create the subscription. This may be a wildcard topic.
scope	object	If the value is non-null, the object specified here becomes the context of the callback function. In other words, when the JavaScript `this` keyword is used in the callback function, it will point to this object. If the value is null, then the object window is used as the default context of the callback function.
onData	function	Callback function for PageBus to invoke when a message is published on the subscribed topic.
subscriberData	any	User-defined. Null values are permitted. This is useful when the subscriber needs to access or update any data members when the callback function is invoked.

For WCC components, the *topic* has the form:

```
com.tibco.wcc.appModelName.prototypeModelName.componentModelName.eventName
```

where:

- `com.tibco.wcc.appModelName` - The application model name (see: `app.getModelName()`)
- `prototypeModelName` - The model name given to the prototype that contains the component, `wccPrototype`, by default.
- `componentModelName` - The model name given to the component, `wccWorkItems`, for example.

When the event to which you've subscribed is published to the PageBus, the following message is returned:

```
{
     jss  : "org.pagebus.msg.Message",
     jssv : "2.0.0",
     head : {topic : eventName},
     body : bodyObject
}
```

where:

- *bodyObject* is a JavaScript object that contains the data expected by the WCC component that subscribes to that event.

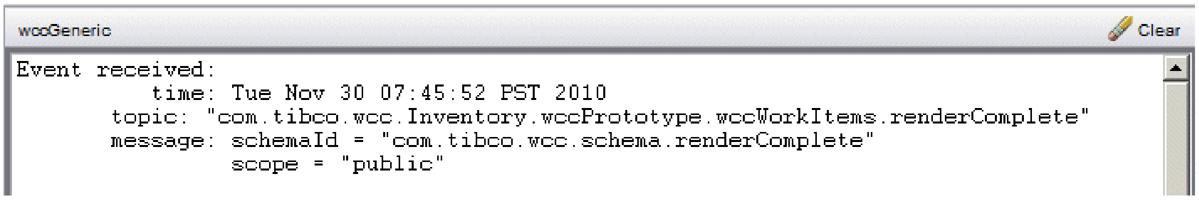
  For information about the data in this object for each event, see Component PageBus Event Payloads.

For more details about using the **subscribe** method, see the *TIBCO PageBus™ Developer's Guide.*

RenderComplete Event

Every component that renders something, like a list, on the screen, issues the **renderComplete** event when it completes the rendering.

Note that this event does NOT appear in the TIBCO General Interface Builder Event Editor as an event you can subscribe to in your application. It does, however, appear in the PageBus Event Monitor component if you include that component in your application:



```
Event received:
          time: Tue Nov 30 07:45:52 PST 2010
       topic: "com.tibco.wcc.Inventory.wccPrototype.wccWorkItems.renderComplete"
       message: schemaId = "com.tibco.wcc.schema.renderComplete"
                scope = "public"
```

There is no payload passed in the **renderComplete** event. It is only an indication that the component (**wccWorkItems** in the example above) has completed rendering.

For more information about the PageBus Event Monitor component, see Viewing Triggered Events using the PageBus Event Monitor Component.

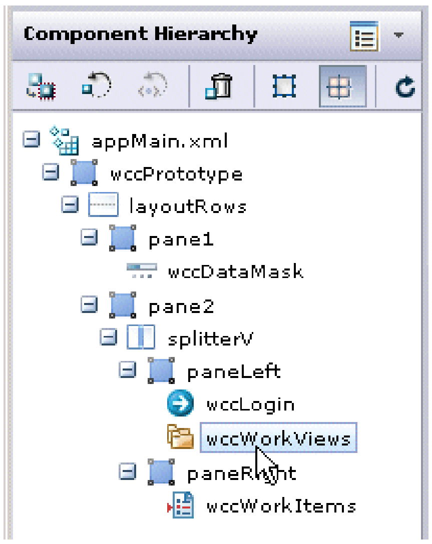
Events Editor

The Events Editor is used to specify the PageBus events to which you would like a WCC component to subscribe.

For information about the PageBus, see Component Events.

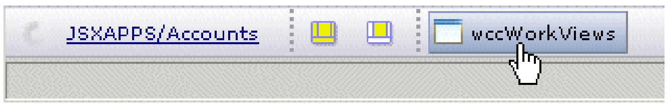
Opening the Events Editor causes it to query all of the Workspace components in the application to determine their public events, and present them in a dialog so that you can choose which to subscribe to. (All public events in the application are shown except for those published by the component you are configuring — it would not make sense for a component to subscribe to its own events.)

To subscribe to events for a particular component, in the **Component Hierarchy** palette, click on the component that will be subscribing to events published by other components:



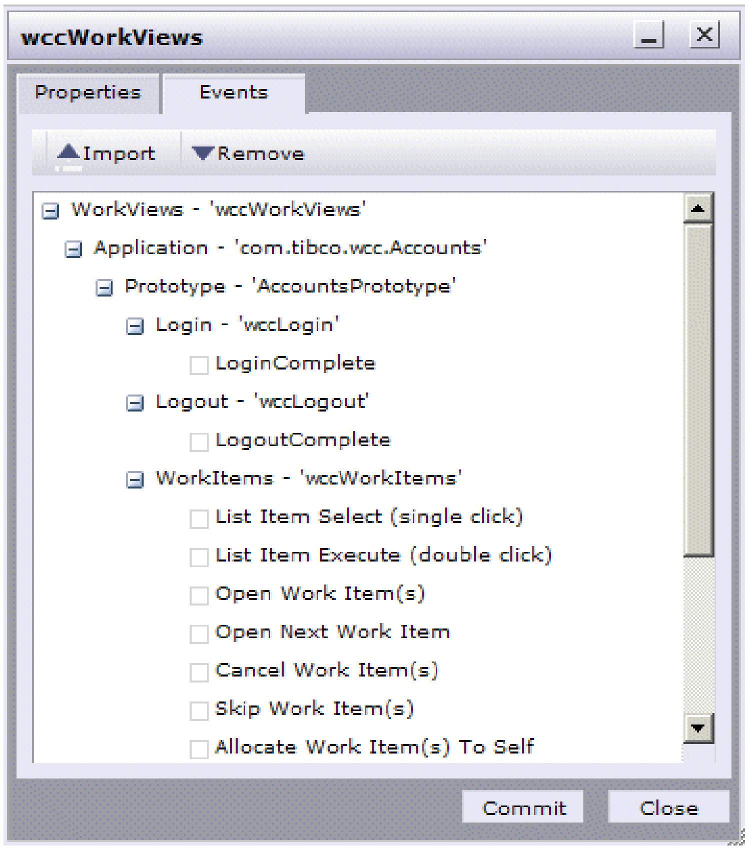
If the Properties/Events Editor is already displayed when you click on the component, it will now show the properties/events for the component you selected in the **Component Hierarchy** palette.

If the Properties/Events Editor is not currently displayed when you click on the component, a button appears in the General Interface Builder taskbar that, when clicked, displays the Properties/Events Editor:



If the Properties/Events Editor is not displayed on the taskbar, double click a component in the **Component Hierarchy** palette to open the editor.

Click the **Events** tab to display the Events Editor. A dialog similar to the following is displayed.



WorkViews - 'wccWorkViews'
Application - 'com.tibco.wcc.Accounts'
Prototype - 'AccountsPrototype'
Login - 'wccLogin'
LoginComplete
Logout - 'wccLogout'
LogoutComplete
WorkItems - 'wccWorkItems'
List Item Select (single click)
List Item Execute (double click)
Open Work Item(s)
Open Next Work Item
Cancel Work Item(s)
Skip Work Item(s)
Allocate Work Item(s) To Self

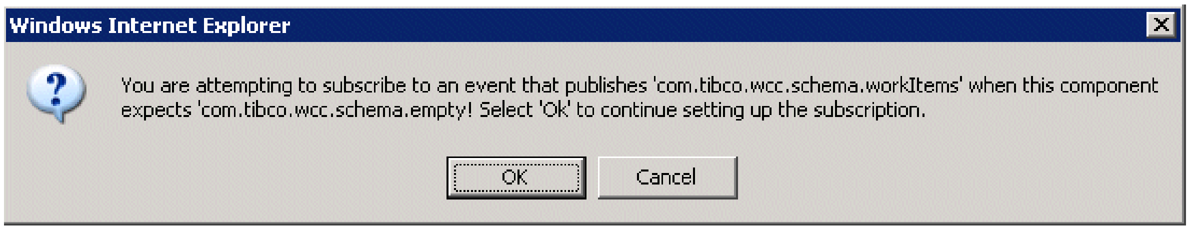
This dialog presents all of the events published by all of the components currently defined in the application (with the exception of the component you are currently configuring). The following describes the information presented on this dialog:

- The first line identifies the component type for which you are configuring subscriptions — “WorkViews” in this example, followed by the name of the particular component — “wccWorkViews” in this example.
- The second line provides the fully qualified application/project name.
- The third line provides the name of the prototype.
- The subsequent information represents components that have been defined in the application, followed by the events that each component is publishing to the PageBus. That is, they are the events to which the component you are configuring can subscribe.

  In the example above, the **wccWorkViews** component (the one we are configuring) can subscribe to events published by the **wccLogin**, **wccLogout**, and **wccWorkItems** components — they are the other components that have been added to the application so far. The **wccLogin** and **wccLogout** components are each publishing one event to which the **wccWorkViews** component can subscribe; the **wccWorkItems** component is publishing a number of events to which the **wccWorkViews** component can subscribe.

Note that although all events published to the PageBus are shown in the Events Editor, it may not make sense to subscribe to some events in the context of the component you are configuring. For instance, when configuring the **wccWorkViews** component, it does not make sense to subscribe to the “List Item Select (single click)**”** event on the **wccWorkItems** component (because it does not provide the information needed to display a list of work views). It does make sense, however, to subscribe to the “LoginComplete” event on the **wccLogin** component to display the work views for the user once the user has successfully logged in.

When you attempt to subscribe to an event, the Event Editor verifies via schema files that the data that will be passed by the event is the data that the component expects. If you attempt to subscribe to an event that publishes a schema that is different than the schema expected by the component, the Event Editor displays a dialog similar to the one below:


Windows Internet Explorer

You are attempting to subscribe to an event that publishes 'com.tibco.wcc.schema.workItems' when this component expects 'com.tibco.wcc.schema.empty! Select 'Ok' to continue setting up the subscription.

In this particular example, the component is expecting an empty schema, meaning all it needs is a valid login.

If the dialog shown above is displayed, the editor will allow you to continue with the subscription if that is what you want.

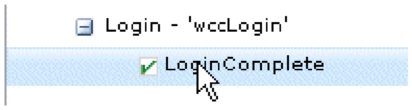
For more information about the individual events for each component, see Component Reference.

Subscribing to Events

The check box to the left of each event name in the Event Editor indicates whether or not the component you are configuring will subscribe to that event.

- If the check box is checked, the component subscribes to that event.
- If the check box is not checked, the component does not subscribe to the event.

To specify that you want the component to subscribe to an event, double-click the desired event name. This causes the event's check box to become checked:



To unsubscribe to an event, double-click the event name again to remove the check mark.

After checking the appropriate event check boxes in the Events Editor, commit your changes by clicking on the **Commit** button. This immediately applies the changes you've made.

Events From External Applications

Because Workspace components need to communicate with applications that exist outside of their own application, the Events Editor provides an *Import* function that is used to import *event definition* files that contain the definitions of events in external applications. This allows you to see, and subscribe to, events in the Events Editor that are being published by external applications.

The WCC Add-in provides an *Application Publish Definer* that is used to create an event definition file. You can then import that file into another WCC application and subscribe to the events published by the external application.

Event definition files must conform to the schema defined in the file:

StudioHome\wcc*version*\JSXAPPS*WCCProjectName*\defs\app.pub.schema.xsd

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.

- *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".

The following sub-topics describe the steps necessary to create the event definition file, then import that file into another WCC application so it can subscribe to the events defined in that file.

Creating the External Application Event Definition

Use the Application Publish Definer to create an event definition file.

Procedure

1. With the external application open in TIBCO General Interface Builder, select **Project Settings** from the **Project** menu.

   The Project Settings dialog is displayed.
2. Click the **WCC Settings** button.
3. Click the **Open Application Publish Definer** button.

   The Create WCC Application File dialog is displayed. This dialog allows you to enter an application model name that will be used in the name of the event definition file, as follows:

   ```
   WCCProjectName.app.pub.xml
   ```

   where *WCCProjectName* is the name of the current WCC project/application.

   It defaults to using the application model name for your current application.
4. Click the **Create** button.

   A dialog is displayed that allows you to point to the prototype file for your external application.
5. Click the **Include** button.

   The **Include Files** window is displayed.
6. Navigate to, and select, the prototype file for your external application. For example:

   `StudioHome\wcc\version\JSXAPPS\WCCProjectName\application\prototypes\appMain.xml`

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.
   - *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".
7. From the Include Files dialog, click the **Include** button.
8. Repeat steps 6 and 7 to include additional files, if desired.
9. Click the **Save** button to save the event definition file, then click **Close** to close the dialog.

   The new event definition file is saved in the following location:

   `StudioHome\wcc\version\JSXAPPS\WCCProjectName\defs\WCCProjectName.app.pub.xml`
10. Click the **Cancel** button to close the Project Settings dialog.

    The new event definition file can now be imported into another WCC application.

Importing the Event Definition File

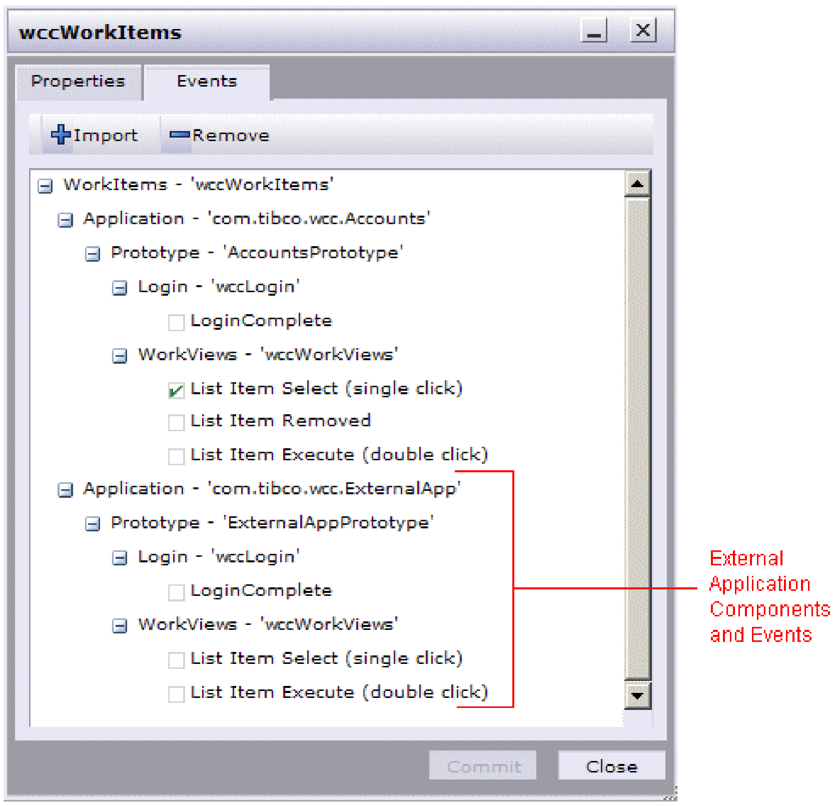
Use the Import function in the Events Editor to import an event definition file.

Procedure

1. Open the custom WCC application from which you want to subscribe to events from an external application.
2. Display the Events Editor (see The Events Editor is used to specify the PageBus events to which you would like a WCC component to subscribe.).
3. Click the **Import** button in the Events Editor.

   The Import Files dialog is displayed.
4. Navigate to the event definition file that was created in the external application (see step 9 in the Creating the External Application Event Definition).
5. Select the event definition file, then click **Import**.

   The Events Editor will now include the components and events from the external application. In the following example, the external application (named "ExternalApp") contains two components whose events your custom application can subscribe:


wccWorkItems
Properties
Events
Import
Remove
WorkItems - 'wccWorkItems'
Application - 'com.tibco.wcc.Accounts'
Prototype - 'AccountsPrototype'
Login - 'wccLogin'
LoginComplete
WorkViews - 'wccWorkViews'
List Item Select (single click)
List Item Removed
List Item Execute (double click)
Application - 'com.tibco.wcc.ExternalApp'
Prototype - 'ExternalAppPrototype'
Login - 'wccLogin'
LoginComplete
WorkViews - 'wccWorkViews'
List Item Select (single click)
List Item Execute (double click)
External Application Components and Events
Commit
Close

6. Subscribe to the desired events from the external application.

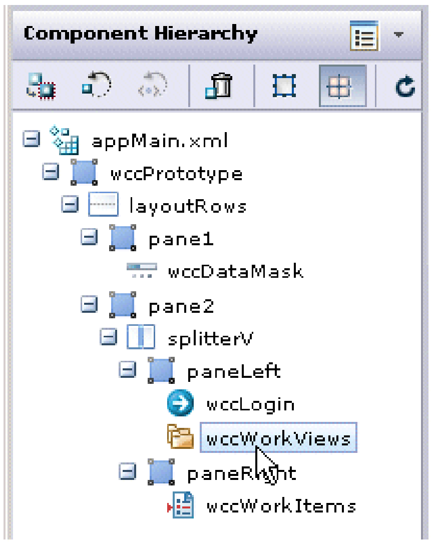
   Note - The external application's components / events can be removed from the Events Editor by clicking on the **Remove** button. This displays the Remove Files dialog from which you can select the desired event definition file for removal.

Properties Editor

The Properties Editor is used to specify property definitions for Workspace components.

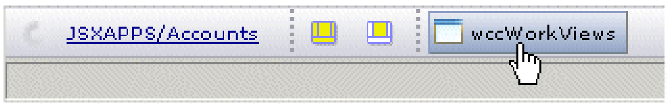
The primary purpose of component properties is to specify how access to functionality exposed by the component is handled. This is done in conjunction with the *user access sets* defined in the custom application's user access configuration file, `userAccess.xml`. User access sets control access in the application based on the logged-in user's *privileges*. For information about user access sets, see *Workspace Configuration and Customization*.

To edit the properties for a particular component, click on the component in the **Component Hierarchy** palette.

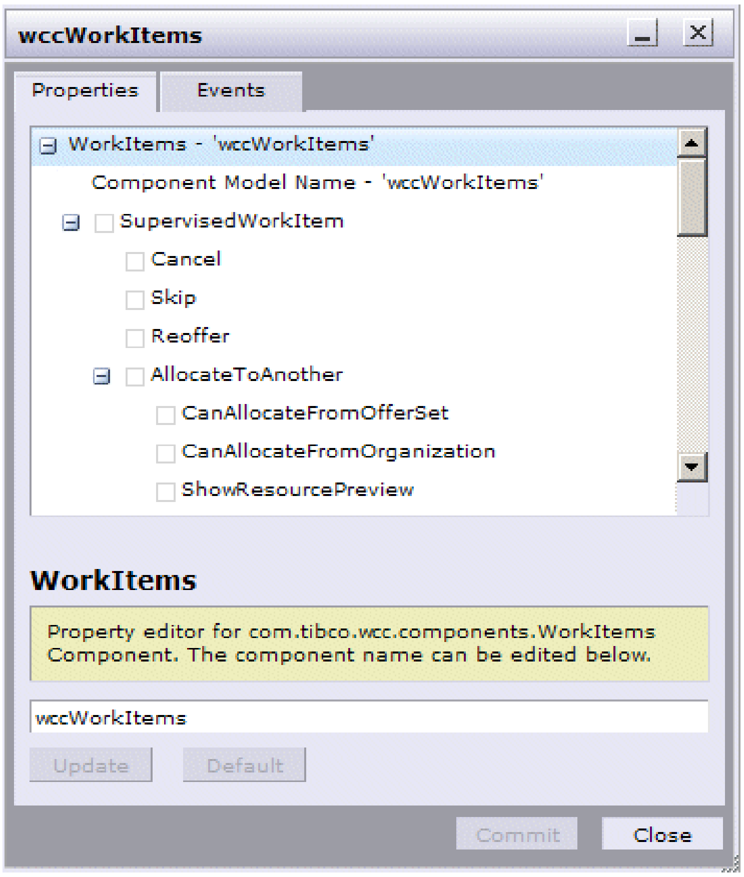


If the Properties/Event Editor is already displayed when you click on the component, it will now show the properties/events for the component you selected in the **Component Hierarchy** palette.

If the Properties/Events Editor is not currently displayed when you click on the component, a button appears in the General Interface Builder taskbar that, when clicked, displays the Properties/Events Editor:



Click on the **Properties** tab to display the Properties Editor.



wccWorkItems

Properties | Events

- WorkItems - 'wccWorkItems'
  - Component Model Name - 'wccWorkItems'
  - ☐ SupervisedWorkItem
    - ☐ Cancel
    - ☐ Skip
    - ☐ Reoffer
    - ☐ AllocateToAnother
      - ☐ CanAllocateFromOfferSet
      - ☐ CanAllocateFromOrganization
      - ☐ ShowResourcePreview

WorkItems

Property editor for com.tibco.wcc.components.WorkItems Component. The component name can be edited below.

wccWorkItems

The Properties Editor provides the following information:

- The first line identifies the component type for which you are configuring properties — "WorkItems" in this example, which represents a work item list. Following the component type is the name of the component.
- The second line identifies the component model name. For more information, see Component Names.
- The subsequent information is a hierarchical tree view of all properties for the component. Each property represents a piece of functionality that is exposed by the component.

  The state of the check box for each property specifies how access to that functionality is handled, as follows:

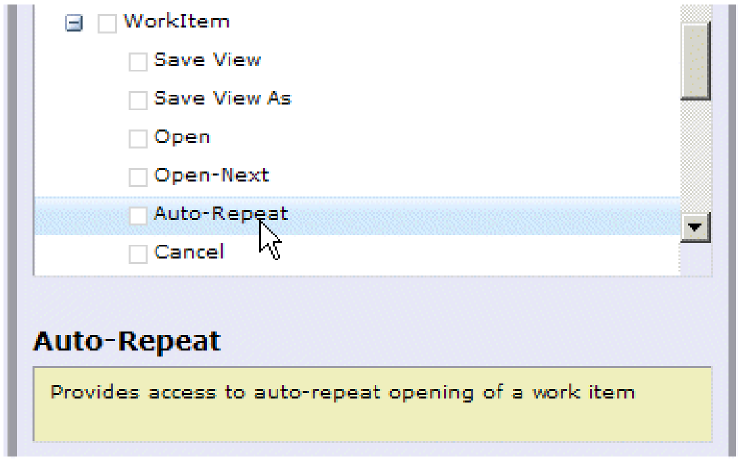
State	Description
Unchecked ☐ Open	Access to the functionality (the **Open** button/menu selection in this example) is controlled totally through *system actions* and *user access sets*(1) .
Checked ✓ Open	Access to the functionality is allowed, even if the setting in the user access sets does not allow it. Note, however, that if a system action denies access to the function, it overrides both the user access set and the property setting for the function on the component.

State	Description
Disabled ☒ Open	Access to the functionality is *not* allowed, event if the function is allowed through a system action and user access set. Note that when you disable access to a parent property in the Properties Editor, all child properties of that parent are also disabled.

(1) For information about system actions and user access sets, see the *TIBCO Workspace Configuration and Customization* guide.

You can change the state of the property check boxes by double-clicking on the property name/check box. It will cycle through the different states — unchecked, checked, and disabled.

Clicking on the property name causes the field below the property list to display a brief description of the functionality defined by that property. For example:



Provides access to auto-repeat opening of a work item

For more information about the specific functionality available for each component, see Component Reference.

Component Names

Each WCC component has both a *name* and a *model name*. The model name is used in the topic of a PageBus event. The PageBus topic consists of a number of 'tokens' separated by a dot '.' character.

For example:

```
com.tibco.wcc.Accounts.AccountsPrototype.wccWorkViews.listItemSelect
```

The tokens are in the following order:

- organization prefix (usually three tokens: always **com.tibco.wcc** for WCC component events)
- application model name (e.g., Accounts)
- prototype model name (e.g., AccountsPrototype)
- component model name (e.g., wccWorkViews)
- event name (e.g., listItemSelect)

  By default, the name and the model name are initially the same. There are times, however, where it make sense to treat several components with unique names as the same source for subscribing to their PageBus events.

  For example, a prototype might contain two work view components, where each one has a unique name. The developer wants the subscribing method to react to events from both components in exactly the same way using a single PageBus subscription. This is accomplished by setting the model name of both components to the same name.

This same concept can be applied to the application model name and the prototype model name.

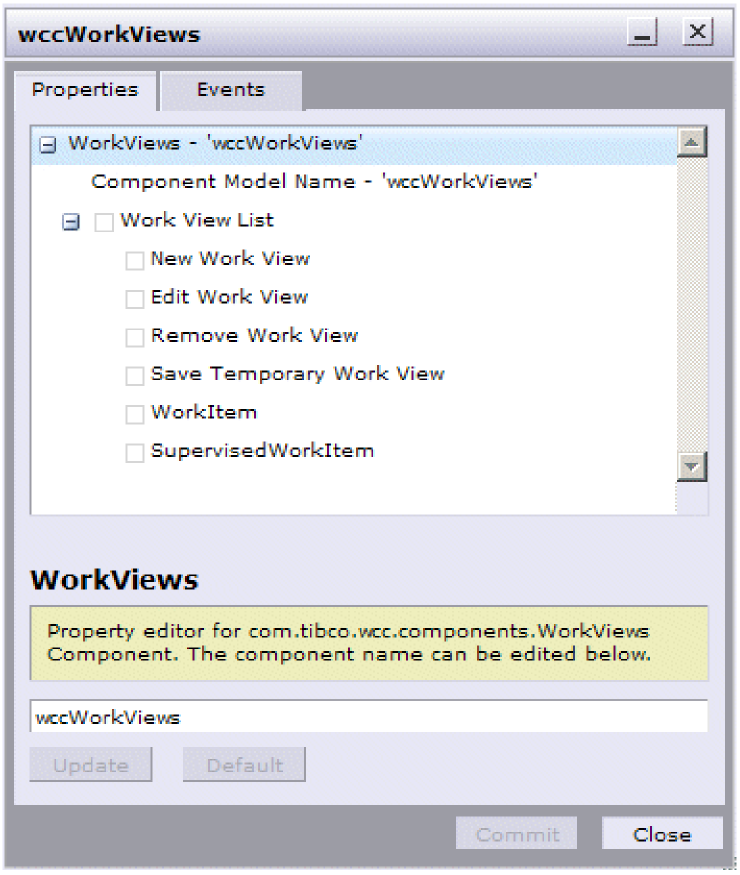
For information about the PageBus, see Component Events.

Changing Component Model Names

The Properties Editor can be used to change both the component name and the component model name.

Changing Component Name

To change the component name, display the Properties Editor and highlight the first line, which shows the component name:



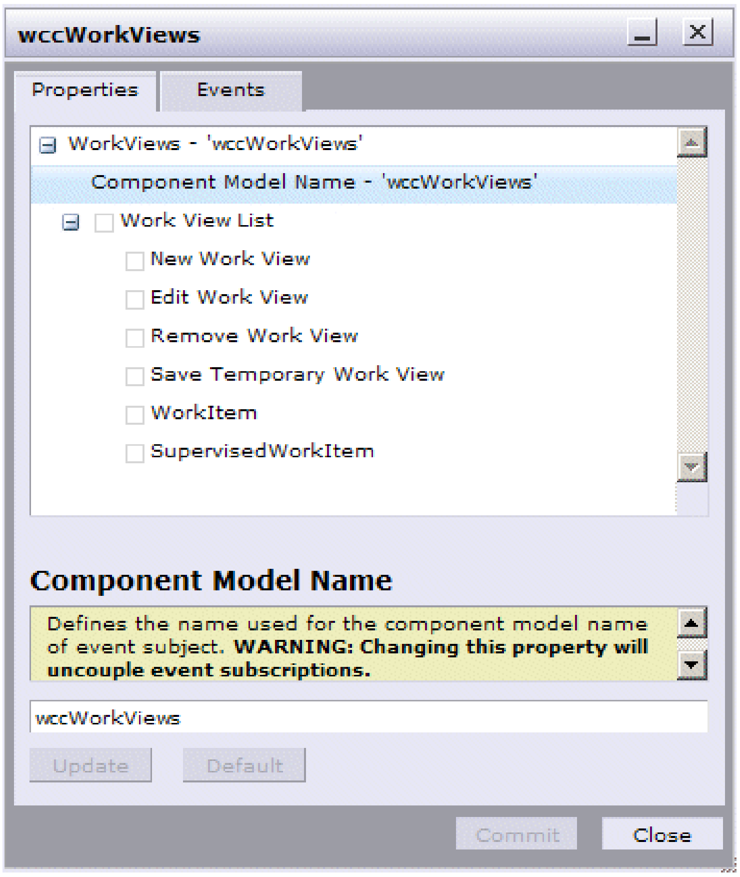
Property editor for com.tibco.wcc.components.WorkViews Component. The component name can be edited below.

wccWorkViews

Enter the desired name in the field near the bottom of the dialog. After making a change to the component name, the **Update** and **Default** buttons become active, allowing you to either save your changes, or to revert back to the default name for the component. The **Commit** button must also be clicked to save all changes made in the Properties Editor.

Changing Component Model Name

To change the component model name, display the Properties Editor and highlight the second line, which shows the component model name:



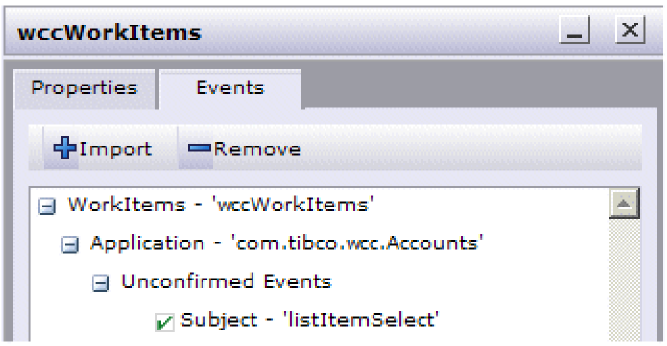
WorkViews - 'wccWorkViews'
Component Model Name - 'wccWorkViews'
Work View List
New Work View
Edit Work View
Remove Work View
Save Temporary Work View
WorkItem
SupervisedWorkItem

Component Model Name

Defines the name used for the component model name of event subject. **WARNING: Changing this property will uncouple event subscriptions.**

wccWorkViews

Note the warning that any events that have already been subscribed to using the model name you are changing will now be uncoupled from the component that had subscribed to it. For instance, if you change a component model name, then, using the Events Editor, look at the component that had subscribed to an event published by the component you changed, it will show the uncoupling, as follows:



WorkItems - 'wccWorkItems'
Application - 'com.tibco.wcc.Accounts'
Unconfirmed Events
✓ Subject - 'listItemSelect'

This is telling you that the **wccWorkItems** component is subscribing to an event that the editor can no longer resolve because the component model name identified in the event topic no longer exists. You can leave it as is if that was your intention, or you can uncheck the unconfirmed event, then subscribe to the appropriate event.

After making a change to the component model name, the **Update** and **Default** buttons become active, allowing you to either save your changes, or to revert back to the default model name for the component. The **Commit** button must also be clicked to save all changes made in the Properties Editor.

Disabling Default Handler

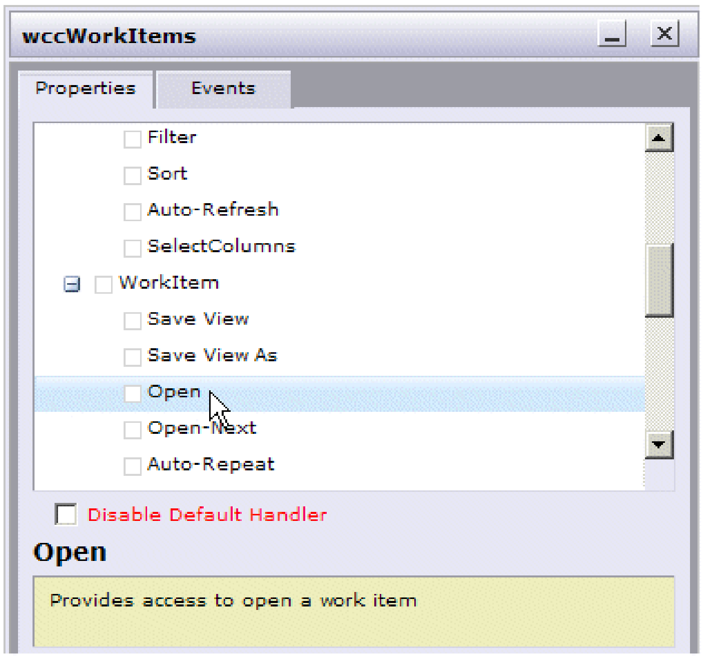
You can disable the default handler for any action that results in an event being triggered.

For example, you may want to perform some sort of custom function when the user clicks on the **Open** button on the work item list. You can disable the normal “open” function, subscribe to the **Open** button event, and perform your custom function — that is, your application must handle the event.

Disabling the default handler can also be used in conjunction with the WCC Application methods to perform a function at a later time. To carry the **Open** button example from above further, you would disable the default handler for the **Open** property on the **WorkItems** component, then subscribe to the **List Item Execute (double click)** event on the **WorkItems** component. When the user double-clicks on a work item, you can perform your custom logic, then call the **openWorkItem** Application method to open the work item. For information about the Application methods you can call to perform a function normally performed by the component, see Application Class Methods.

Procedure

1. Open the Properties Editor for the component that includes the property corresponding to the functionality whose event you want to disable.
2. Click on the property that corresponds to the function. For example:



Provides access to open a work item

   Notice that when you click on a property that has a corresponding event, the **Disable Default Handler** check box appears on the Properties Editor dialog.
3. Click in the **Disable Default Handler** check box.
4. Click **Commit** to save your changes.

   Now when the **Open** work item event fires, the normal open work item function will not take place.

Component Reference

This guide does not provide detailed information about using the functions provided by WCC components. For those details, see the *TIBCO Workspace User's Guide*.

Composite Components

- **Organization Resource Browser** - composite of the Organization Browser and Organization Resource List components.
- **Process Instances Preview** - composite of the Process Instances component and the preview pane(1).
- **Work Items Preview** - composite of the Work Items component and the preview pane.
- **Data View Results** - composite of the Data View[3] List and a pane that contains tabs for viewing the global case data for the selected case reference, as well as work items, process instances, and events associated with the case data.

Data Mask Component

- **Data Mask Component** - block that displays "Loading Data…" mask while the application is retrieving data from the server.

List Container Components

- **Event Views** - list of event views created by the user.
- **Event Viewer** - list of events that are in the currently selected event view.
- **Process Templates** - list of process templates available on the system. This component is used by the Start Instance Component to display the list of processes the user can start an instance of. It provides a flat list of available process templates.
- **Process Templates Ex** - list of process templates available on the system. This component is used by the Create / Edit Process View wizard to display the list of processes that the user can include in the process view. This component provides a tree structure with expandable / collapsible nodes, each node representing a process template. Expanding a node causes all versions of the process template to be displayed below the process template name.
- **Process Views** - list of views of process instances, for a set of process templates, created by the user.
- **Process Instances** - list of all process instances in the currently selected process view (does not include the preview pane — for a preview pane, see the **Process Instances Preview** composite component).
- **Work Views** - list of views of work items created by the user.
- **Work Items** - list of work items in the selected work view (does not include the preview pane — for a preview pane, see the **Work Items Preview** composite component).
- **Business Services** - list of business services available to the user.
- **Data Views** - list of data views(1) created by the user.
- **Data View List** - list of case references for the selected data view(1).

[2] The preview pane is viewed as a separate component in the context of composite components, although it cannot be installed separately.

[3] As of version 4.0 of Workspace, "data views" are now called "case views". Although, in General Interface Builder, the term "data view" is still used, the component appears as a "case view" in the application.

Login Component

- **Login Component** - interface for performing application login.

LocaleSelector

- **LocaleSelector Component** - interface for changing the language.

Toolbar Button Component

- **Toolbar Button Component** - used to display one of the following buttons in your WCC application:
  - Organization Browser Window Button - displays the Configuration Administrator dialog, which is used to configure deployed applications.
  - Logout Button - logs the user out of the application.
  - Open Next Work Item Button - opens the next available work item from the work item list.
  - Options Button - displays the Options dialog, which is used to establish default settings for the currently logged-in user.
  - Organization Browser Window Button - displays the Organization Resource Browser component, which is a composite component used to browse the organization model, create LDAP containers, map users to groups/positions, etc.

Organization Browser Components

- **Organization Browser** - displays a graphical representation of the organization model. Also allows the user to create LDAP containers, which contain resources that can be mapped to groups and positions in the organization model.
- **Organization Resource List** - displays a list of resources that have been added to the selected LDAP container, or that have been mapped to the selected organizational entity.

Start Instance Component

- **Start Instance Component** - allows you to select a process template and start an instance of it.

Viewing Triggered Events using the PageBus Event Monitor Component

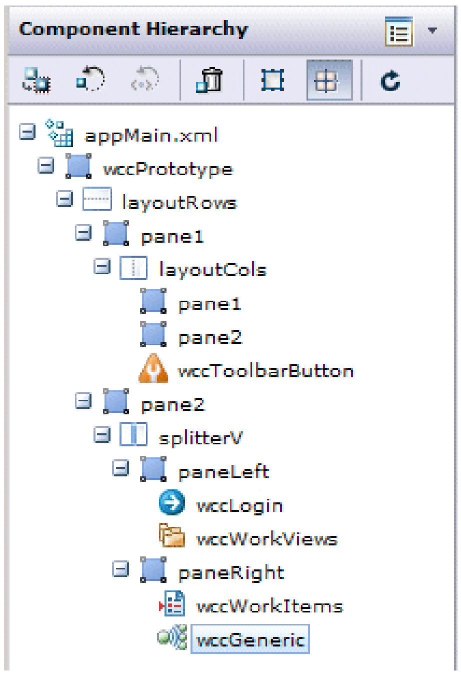
The PageBus Event Monitor component is available as a troubleshooting tool. It allows you to view information about events as they are triggered, giving you a visual confirmation that events are occurring as you test your application.

To use the PageBus Event Monitor component, follow these steps (note that this procedure assumes you have a custom application using WCC components already created — the steps below use the custom application created with the How to Create a BPM Desktop Using Components Tutorial tutorial on How to Create a BPM Desktop Using Components Tutorial.):

Procedure

1. From the **Component Libraries** palette, select and drag the **PageBus Event Monitor** component onto a pane in the **Component Hierarchy** palette.

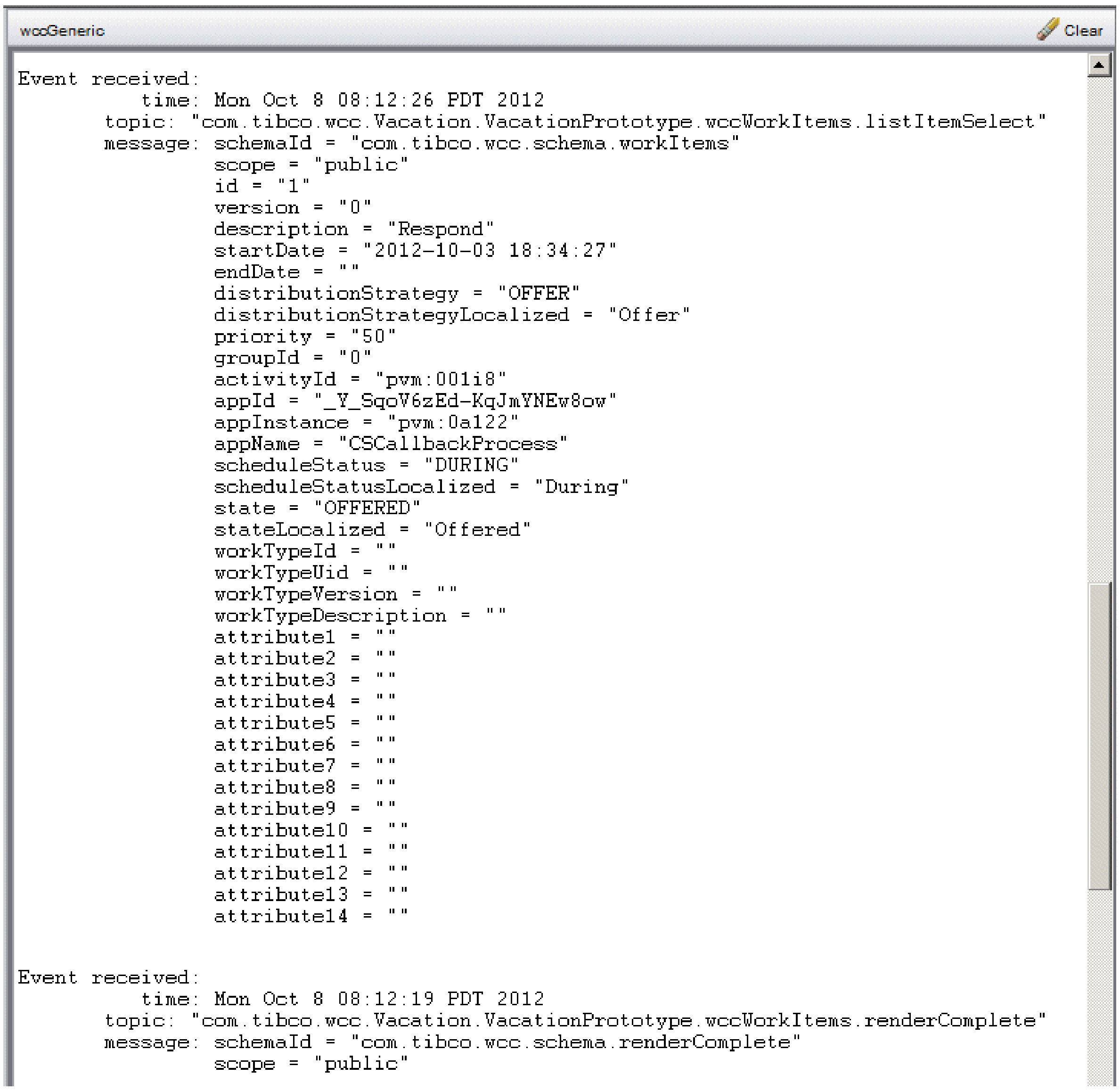
   For example:



Note that the PageBus Event Monitor component appears as "wccGeneric" in the Component Hierarchy.

There is no need to explicitly subscribe to events from the PageBus Event Monitor component — it automatically subscribes to events published by the components that have been added to the application (this can be configured; see Specifying the Events to Which the PageBus Event Monitor Component Subscribes).

2. Click **Commit** to save the changes.
3. Select **Save** from the **File** menu.
4. In Windows Explorer, navigate to your workspace directory and execute the `ProjectName.html` file, where *ProjectName* is the name you gave your project in General Interface Builder. (Note that you can also test/run the application from within General Interface Builder, rather than via Windows Explorer.)
5. Perform some actions in your application, then view the output caused by the **PageBus Event Monitor** component. An example is shown below:



```
Event received:
          time: Mon Oct 8 08:12:26 PDT 2012
       topic: "com.tibco.wcc.Vacation.VacationPrototype.wccWorkItems.listItemSelect"
       message: schemaId = "com.tibco.wcc.schema.workItems"
                scope = "public"
                id = "1"
                version = "0"
                description = "Respond"
                startDate = "2012-10-03 18:34:27"
                endDate = ""
                distributionStrategy = "OFFER"
                distributionStrategyLocalized = "Offer"
                priority = "50"
                groupId = "0"
                activityId = "pvm:001i8"
                appId = "_Y_SqoV6zEd-KqJmYNEw8ow"
                appInstance = "pvm:0a122"
                appName = "CSCallbackProcess"
                scheduleStatus = "DURING"
                scheduleStatusLocalized = "During"
                state = "OFFERED"
                stateLocalized = "Offered"
                workTypeId = ""
                workTypeUid = ""
                workTypeVersion = ""
                workTypeDescription = ""
                attribute1 = ""
                attribute2 = ""
                attribute3 = ""
                attribute4 = ""
                attribute5 = ""
                attribute6 = ""
                attribute7 = ""
                attribute8 = ""
                attribute9 = ""
                attribute10 = ""
                attribute11 = ""
                attribute12 = ""
                attribute13 = ""
                attribute14 = ""


Event received:
          time: Mon Oct 8 08:12:19 PDT 2012
       topic: "com.tibco.wcc.Vacation.VacationPrototype.wccWorkItems.renderComplete"
       message: schemaId = "com.tibco.wcc.schema.renderComplete"
                scope = "public"
```

The **PageBus Event Monitor** component output provides information about each event to which you've subscribed that is caused by an action in your application. It includes the event topic string, as well as the message sent to the PageBus.

Note that any JavaScript objects in the message payload are converted to JSON strings before being displayed in the **PageBus Event Monitor** component. For information about the data included in the payload for each event/schema, see Component PageBus Event Payloads.

Use the **Clear** button to clear the current information, but continue displaying the **PageBus Event Monitor** component output window.

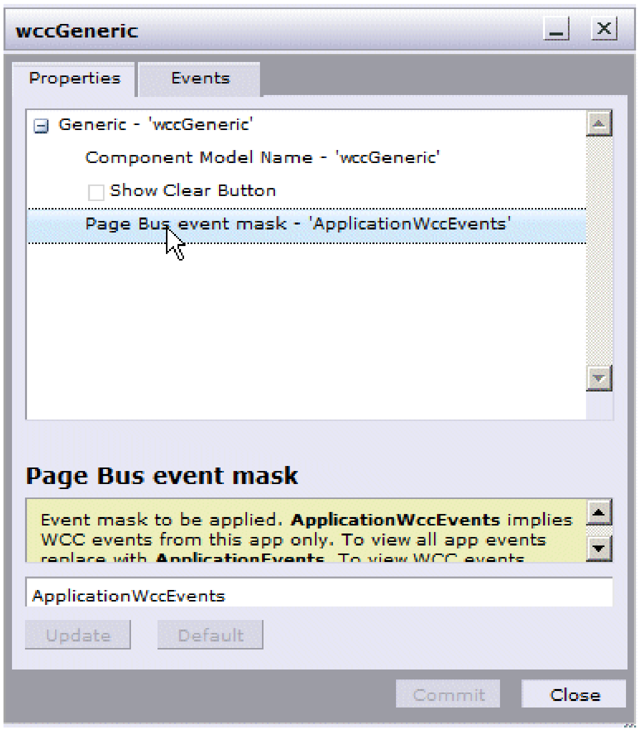
The **PageBus Event Monitor** component has a single property — **Show Clear Button** — that controls the visibility of the **Clear** button. It also publishes a single event — **Generic Clear** — that fires when the **Clear** button is clicked.

Specifying the Events to Which the PageBus Event Monitor Component Subscribes

You can configure the PageBus Event Monitor component to subscribe to a specific set of events.

Procedure

1. With your WCC application open in TIBCO General Interface Builder, select the PageBus Event Monitor component (wccGeneric) in the Component Hierarchy.
2. Display the Properties/Event editor by clicking on the **wccGeneric** button in the TIBCO General Interface Build task bar.
3. On the **Properties** tab, select **Page Bus event mask**:



Generic - 'wccGeneric'
Component Model Name - 'wccGeneric'
Show Clear Button
Page Bus event mask - 'ApplicationWccEvents'

Page Bus event mask

Event mask to be applied. **ApplicationWccEvents** implies WCC events from this app only. To view all app events replace with **ApplicationEvents**. To view WCC events

ApplicationWccEvents

4. In the field below the **Page Bus event mask** section, enter one of the following strings (without quotes) to specify which events the PageBus Event Monitor component should subscribe (note that the text above the field also describes this setting):

   - **ApplicationWccEvents** - The PageBus Event Monitor component will subscribe to events from WCC components in the current application only.
   - **ApplicationEvents** - The PageBus Event Monitor component will subscribe to events from all components in the current application only.
   - **AllApplicationsWccEvents** - The PageBus Event Monitor component will subscribe to events from WCC components from all applications.
   - **AllApplicationsEvents** - The PageBus Event Monitor component will subscribe to events from all components from all applications.

     As shown in the illustration above, the default is to subscribe only to events published by WCC components in the current application.

5. After entering the event mask, click **Commit** to save the new setting.

Composite Components

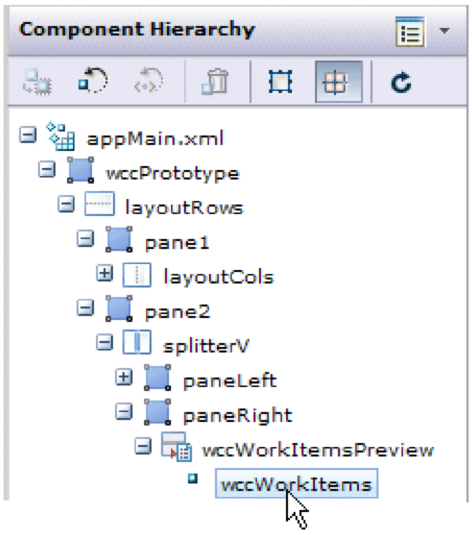
There are *composite components* available that combine more than one component into a single component.

The available composite components are:

- **Organization Resource Browser** - composite of the Organization Browser and Organization Resource List components.
- **Process Instances Preview** - composite of the Process Instances component and the preview pane.
- **Work Items Preview** - composite of the Work Items component and the preview pane.
- **Data View Results** - composite of the Data View List component[4] and a pane that contains tabs for viewing the global case data for the selected case reference, as well as work items, process instances, and events associated with the case data.

Composite Component Properties

When setting properties on a composite component, you must drill down to the underlying component itself:



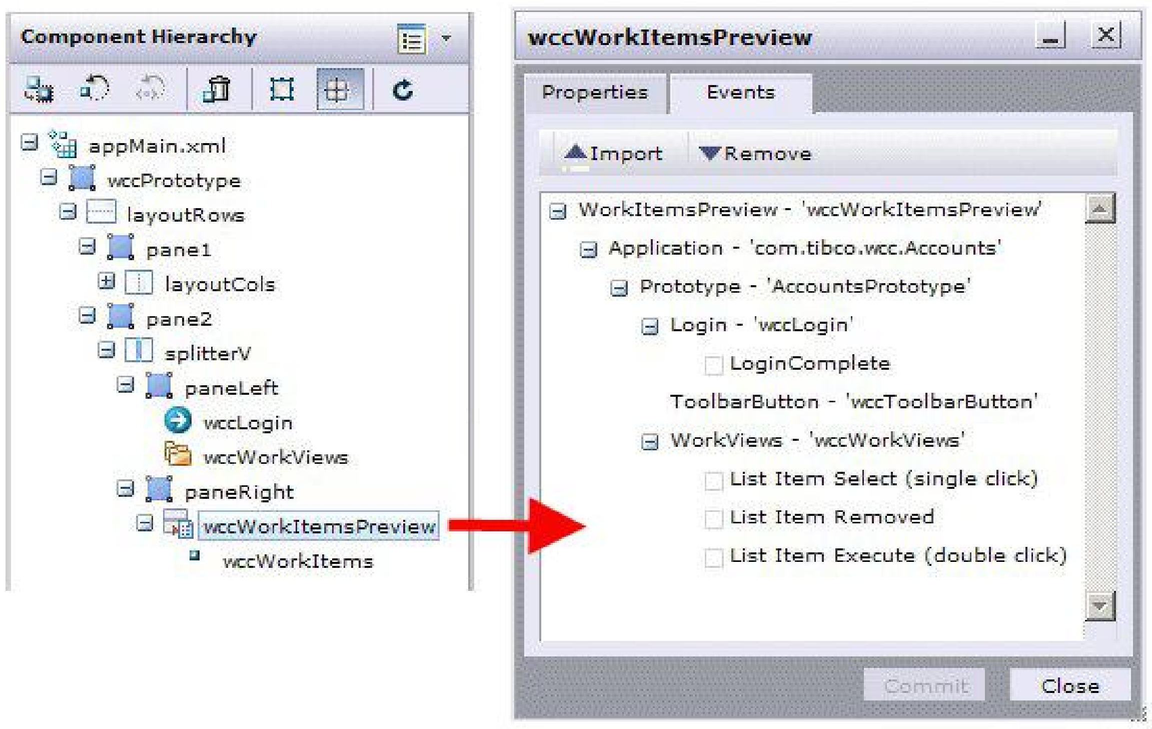
Properties are set on the underlying component, not on the composite component.

The **Data View Results** composite is an exception; you can set properties on the Data View Results component. For more information, see Data View Results Component Properties .

Composite Component Events

A composite component can subscribe to events published by other components.

[4] As of version 4.0 of Workspace, "data views" are now called "case views". Although, in General Interface Builder, the term "data view" is still used, the component appears as a "case view" in the application.



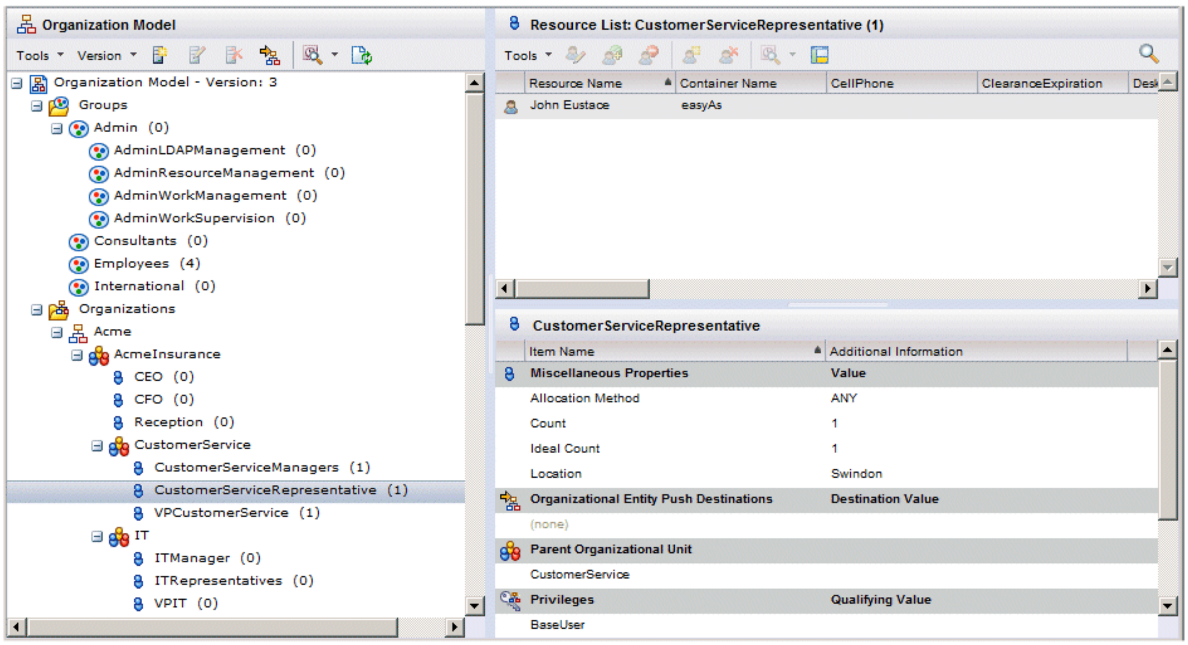
However, the underlying components cannot subscribe to events.

Also, the composite component itself does not publish events — the underlying components do. Therefore, if another component in your application wants to subscribe to events from the composite, it must actually subscribe to events from the underlying component (note that the preview pane does not publish events).

Organization Resource Browser

The **Organization / Resource Browser** component is a composite of the **Organization Browser** component and the **Organization Resource** List component.

This component displays a dialog similar to the following:



Resource Name	Container Name	CellPhone	ClearanceExpiration	Desk
John Eustace	easyAs			

Item Name	Additional Information
Miscellaneous Properties	**Value**
Allocation Method	ANY
Count	1
Ideal Count	1
Location	Swindon
Organizational Entity Push Destinations	**Destination Value**
(none)	
Parent Organizational Unit	
CustomerService	
Privileges	**Qualifying Value**
BaseUser	

The organization model is displayed in the left pane. Note, however, that a resource list is not displayed in the right pane until the user selects an organizational entity or LDAP container from the left pane.

Also see Organization Browser Window Button for information about placing a button in your application that displays this component.

Subscribe To:

To display the **Organization / Resource Browser** component, subscribe to the following:

- **Login** component:
  - "Login Complete" event

Organization / Resource Browser Component Properties

None — composite components do not have their own properties. To set the properties for the individual components that are part of the composite, click on the underlying component in the **Component Hierarchy**, then use the Properties Editor to set access for functions provided by that component.

For information about the properties available for the components included with **Organization / Resource Browser**, see:

- Organization Browser Component Properties
- Organization Resource List Component Properties

Organization / Resource Browser Component Events

None — composite components do not publish events themselves, although the individual components that comprise the composite publish events. Therefore, other components cannot subscribe to events of a composite, but they can subscribe to events of the underlying components.

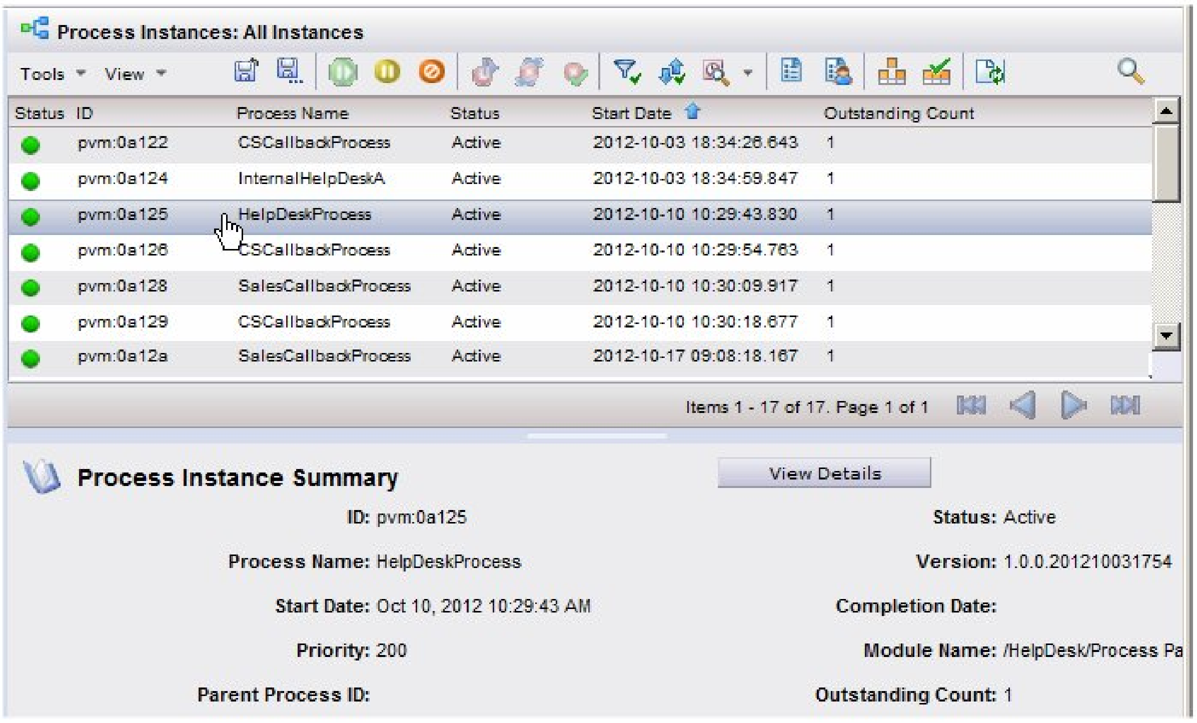
For information about the events published by the components included with **Organization / Resource Browser**, see:

- Organization Browser Component Events
- Organization Resource List Component Events

Process Instances Preview

The **Process Instances Preview** component is a composite of the process instance list component (**Process Instances**) and the preview pane. (This is considered a composite, although technically it does not comprise two components that you can see in the **Component Hierarchy**; in this context, the preview pane is considered a component, although it cannot be installed separately.)

When a process instance is *selected* (single click) in the process instance list displayed by this component, the process instance summary is displayed in the preview pane:


Process Instances: All Instances
Tools
View
Items 1 - 17 of 17. Page 1 of 1
View Details

Status	ID	Process Name	Status	Start Date	Outstanding Count
●	pvm:0a122	CSCallbackProcess	Active	2012-10-03 18:34:26.643	1
●	pvm:0a124	InternalHelpDeskA	Active	2012-10-03 18:34:59.847	1
●	pvm:0a125	HelpDeskProcess	Active	2012-10-10 10:29:43.830	1
●	pvm:0a126	CSCallbackProcess	Active	2012-10-10 10:29:54.763	1
●	pvm:0a128	SalesCallbackProcess	Active	2012-10-10 10:30:09.917	1
●	pvm:0a129	CSCallbackProcess	Active	2012-10-10 10:30:18.677	1
●	pvm:0a12a	SalesCallbackProcess	Active	2012-10-17 09:08:18.167	1

Process Instance Summary

ID: pvm:0a125	**Status:** Active
Process Name: HelpDeskProcess	**Version:** 1.0.0.201210031754
Start Date: Oct 10, 2012 10:29:43 AM	**Completion Date:**
Priority: 200	**Module Name:** /HelpDesk/Process Pa
Parent Process ID:	**Outstanding Count:** 1

The process instance list that is displayed using the **Process Instances Preview** component also contains the **Preview** icon, as well as the **Preview** menu selections on the **View** menu to allow the user to turn off the preview pane, if desired.

Subscribe To:

To display the **Process Instances Preview** component, subscribe to one of the following:

- **Process Views** component:
  - List Item Select (single click) event
  - List Item Execute (double click) event

Process Instances Preview Component Properties

None — composite components do not have their own properties. To set the properties for the individual components that are part of the composite, click on the underlying component in the **Component Hierarchy**, then use the Properties Editor to set access for functions provided by that component.

For information about the properties available for the components included with **Process Instances Preview**, see:

- Process Instances Component Properties

Process Instances Preview Component Events

None — composite components do not publish events themselves, although the individual components that comprise the composite publish events. Therefore, other components cannot subscribe to events of a composite, but they can subscribe to events of the underlying components.

For information about the events published by the components included with **Process Instances Preview**, see:

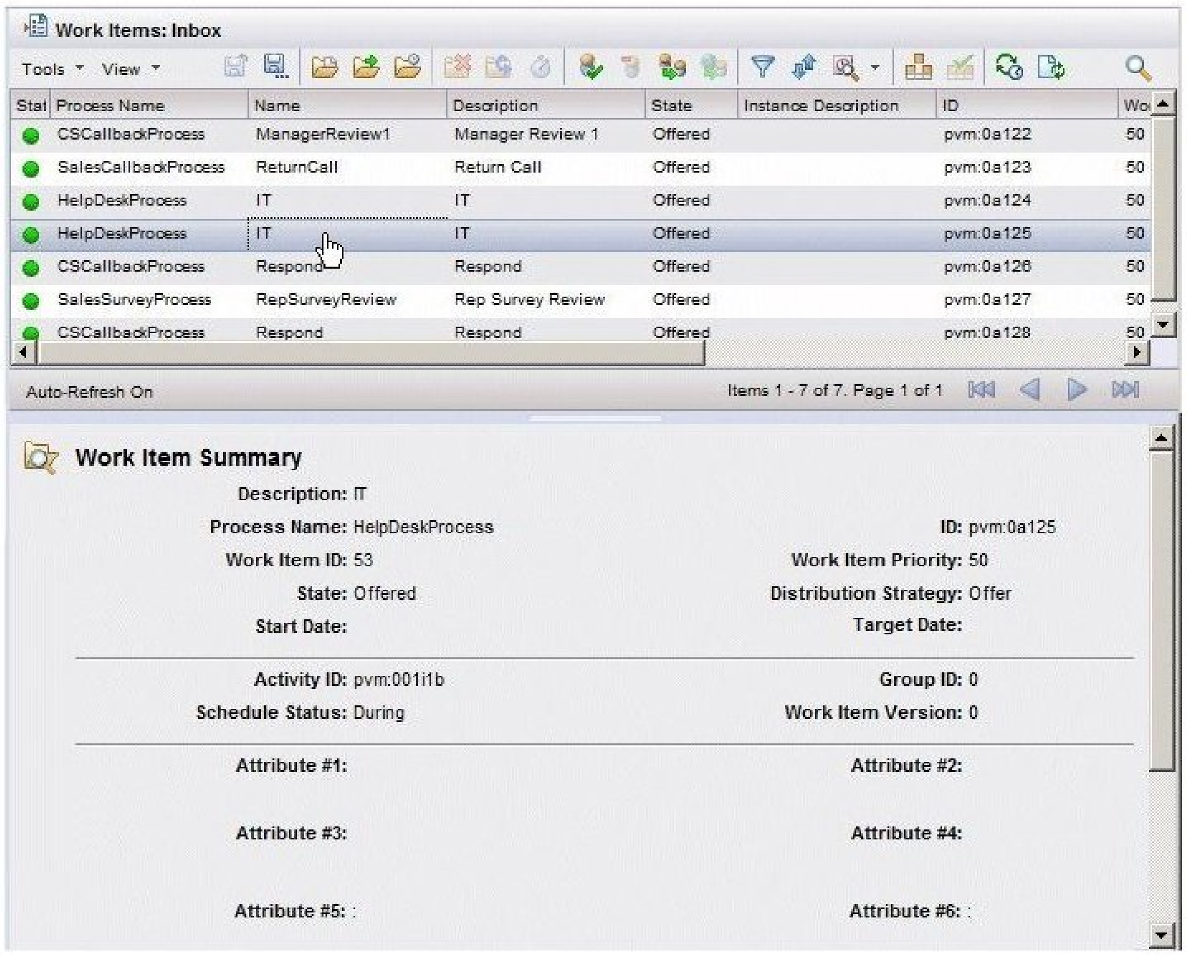
- Process Instances Component Events

Work Items Preview

The **Work Items Preview** component is a composite of the work item list component (**Work Items**) and the preview pane. (This is considered a composite, although technically it does not comprise two components that you can see in the **Component Hierarchy**; in this context, the preview pane is considered a component, although it cannot be installed separately.)

When a work item is selected (by single clicking) in the work item list displayed by this component, the work item summary is displayed in the preview pane. When a work item is opened (by double clicking, or selecting the **Open Selected Work Items** function), the work item form is displayed in the preview pane.

The **Work Items Preview** component displays the work item list, followed by the preview pane:



Stat	Process Name	Name	Description	State	Instance Description	ID	Wo
●	CSCallbackProcess	ManagerReview1	Manager Review 1	Offered		pvm:0a122	50
●	SalesCallbackProcess	ReturnCall	Return Call	Offered		pvm:0a123	50
●	HelpDeskProcess	IT	IT	Offered		pvm:0a124	50
●	HelpDeskProcess	IT	IT	Offered		pvm:0a125	50
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a126	50
●	SalesSurveyProcess	RepSurveyReview	Rep Survey Review	Offered		pvm:0a127	50
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a128	50

Auto-Refresh On   Items 1 - 7 of 7. Page 1 of 1

Work Item Summary

Description: IT

Process Name: HelpDeskProcess   ID: pvm:0a125

Work Item ID: 53   Work Item Priority: 50

State: Offered   Distribution Strategy: Offer

Start Date: Target Date:

Activity ID: pvm:001i1b   Group ID: 0

Schedule Status: During   Work Item Version: 0

Attribute #1:   Attribute #2:

Attribute #3:   Attribute #4:

Attribute #5: :   Attribute #6: :

The work item list that is displayed using the **Work Items Preview** component also contains the **Preview** icon, as well as the **Preview** menu selections on the **View** menu to allow the user to display the work item form in a floating window. (Note that the **Work Items** component, which also displays the work item list, only opens the work item form in a floating window — it does not have the **Preview** icon nor the **Preview** menu selections on the **View** menu.)

Subscribe To:

To display the **Work Items Preview** component, subscribe to one of the following:

- **Work Views** component:

- List Item Select (single click) event
- List Item Execute (double click) event

Work Items Preview Component Properties

None — composite components do not have their own properties. To set the properties for the individual components that are part of the composite, click on the underlying component in the **Component Hierarchy**, then use the Properties Editor to set access for functions provided by that component.

For information about the properties available for the components included with **Work Items Preview**, see:

- Work Items Component Properties

Work Items Preview Component Events

None — composite components do not publish events themselves, although the individual components that comprise the composite publish events. Therefore, other components cannot subscribe to events of a composite, but they can subscribe to events of the underlying components.

For information about the events published by the components included with **Work Items Preview**, see:

- Work Items Component Events

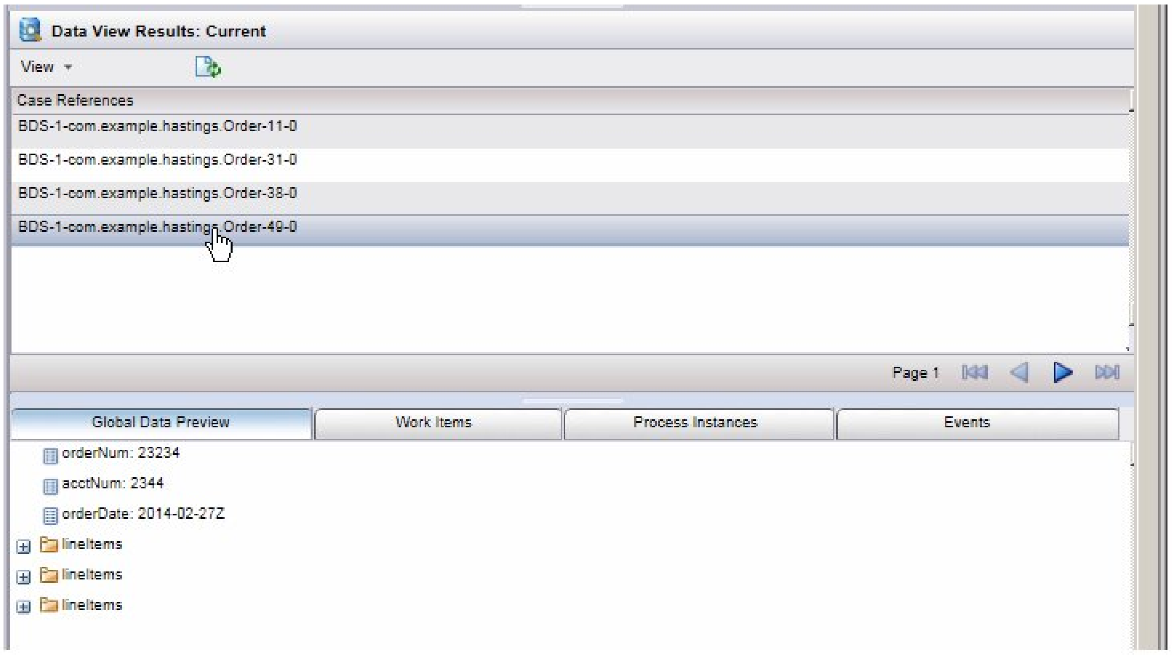
Data View Results

The **Data View Results** component is a composite of the data view list component (**Data View List**) and a pane that contains tabs for viewing the global case data for the selected case reference, as well as work items, process instances, and events associated with the case data.

As of version 4.0 of Workspace, "data views" are now called "case views". Although, in General Interface Builder, the term "data view" is still used, the component appears as a "case view" in the application.

(This is considered a composite, although technically it does not comprise two components that you can see in the **Component Hierarchy**; in this context, the pane containing case details is considered a component, although it cannot be installed separately.)

When a case reference is *selected* (single click) in the case view list displayed by this component, the case details are displayed in the lower-right pane:


Data View Results: Current
View
Case References
BDS-1-com.example.hastings.Order-11-0
BDS-1-com.example.hastings.Order-31-0
BDS-1-com.example.hastings.Order-38-0
BDS-1-com.example.hastings.Order-49-0
Page 1
Global Data Preview
Work Items
Process Instances
Events
orderNum: 23234
acctNum: 2344
orderDate: 2014-02-27Z
lineItems
lineItems
lineItems

Subscribe To:

To display the **Data View Results** component, subscribe to one of the following:

- **Data Views** component:
  - List Item Select (single click) event
  - List Item Execute (double click) event

Data View Results Component Properties

Note that most composite components do not have their own properties. The **Data View Results** composite component is an exception.

The **Data View Results** component contains the following properties, which are used to control access to each of the tabs available from the case data details pane (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
Global Data Preview	Controls access to the **Global Data Preview** tab on the case data details pane.
Work Items List	Controls access to the **Work Items** tab on the case data details pane.
Process Instance List	Controls access to the **Processs Instances** tab on the case data details pane.
Event Viewer	Controls access to the **Events** tab on the case data details pane.

Data View Results Component Events

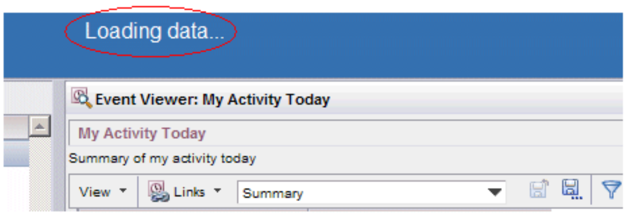
None — composite components do not publish events themselves, although the individual components that comprise the composite publish events. Therefore, other components cannot subscribe to events of a composite, but they can subscribe to events of the underlying components.

For information about the events published by the components included with the **Data View Results** component, see:

- Data View List Component Events
- Work Items Component Events
- Process Instances Component Events
- Event Viewer Component Events

Data Mask Component

The **Data Mask** component provides a “Loading data ...” data mask while the application is retrieving information from the server and rendering it on the screen.



This provides a visual to the user that the application is working.

Note that the "Loading Data ..." text is white. It must be on a colored background to be able to see it.

Also, the **Data Mask** component does not have any editable features — i.e., no properties nor events. It also does not require that data be passed to it, that is, it does not need to subscribe to an event of another component.

List Container Components

The list container components display various lists in your custom application.

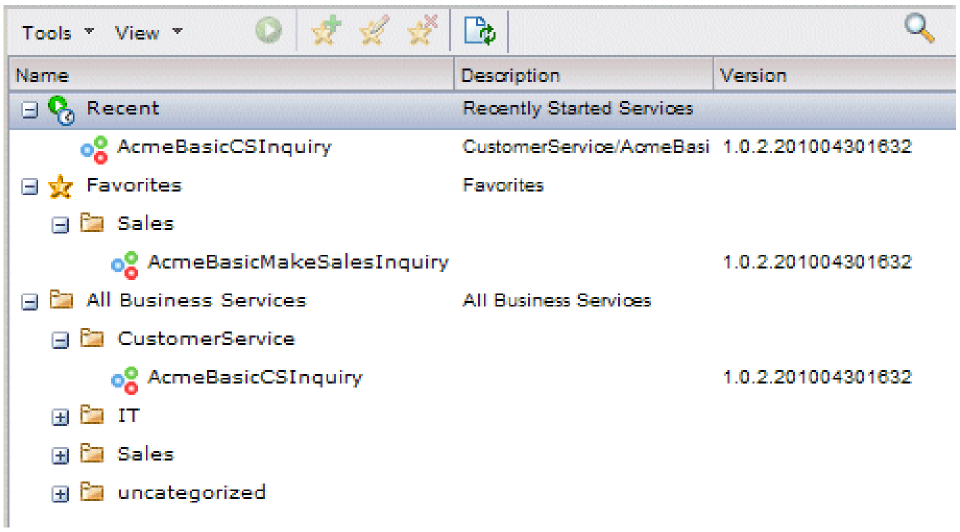
They are:

- **Business Services** - list of all of the business services that the logged-in user is authorized to start.
- **Event Views** - list of event views created by the user.
- **Event Viewer** - list of events in the currently selected event view.
- **Process Templates** - list of available process templates (in a flat list structure) that is filterable and sortable.
- **Process Templates Ex** - list of available process templates (in a tree structure).
- **Process Views** - list of process views created by the user.
- **Process Instances** - list of process instances in the selected process view.
- **Work Views** - list of work views created by the user.
- **Work Items** - list of work items in the selected work view.
- **Data Views** - list of case views[5] created by the user.
- **Data View List** - list of cases that are the result of the view selected in the case views(1) list.

Business Services

The **Business Services** component displays a list of all of the business services that the logged-in user is authorized to start.

An example business service list is illustrated below:



Name	Description	Version
Recent	Recently Started Services	
AcmeBasicCSInquiry	CustomerService/AcmeBasi	1.0.2.201004301632
Favorites	Favorites	
Sales		
AcmeBasicMakeSalesInquiry		1.0.2.201004301632
All Business Services	All Business Services	
CustomerService		
AcmeBasicCSInquiry		1.0.2.201004301632
IT		
Sales		
uncategorized		

For details about the functions available from the business service list, see the "Business Services" topic in the *TIBCO Workspace User's Guide*.

Access to each of the functions available in the business service list is controlled by properties on the **Business Services** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Business Services Component Properties.

The **Business Services** component also publishes events for actions executed from the business service list. For information about these events, see Business Services Component Events.

Subscribe To:

To display the **Business Services** component, subscribe to the following event:

- **Login** component:
  - "Login Complete" event

Business Services Component Properties

The **Business Services** component contains the following properties, which are used to control access to each of the functions available from the business service list (these properties are used in conjunction with "user access sets" to control access — for more information, see The Properties Editor is used to specify property definitions for Workspace components.).

Property	Description
BusinessServices List	Enables/disables the ability to view the business service list.

[5] As of version 4.0 of Workspace, "data views" are now called "case views". However, in General Interface Builder, the term "data view" is still used, the component appears as a "case view" in the application.

Property	Description
Start Business Service	Enables/disables the ability to start a business service. Controls access to the **Start** button on the toolbar, as well as the **Start Business Service** selection on the business service list **Tools** menu.
Favorites	Enables/disables the ability to add business services to the **Favorites** section in the business service list. Controls access to the **Add to Favorites** button on the toolbar, as well as the **Add to Favorites** selection on the business service list **Tools** menu.

Business Services Component Events

The **Business Services** component publishes the following events, which fire when the action described by the event occurs.

Event	Description
List Item Select (single click)	Indicates the user has *selected* a business service in the business service list. Fires when the user single-clicks a business service *or* a category in the business service list, or when the user moves the highlight bar on the business service list using the keyboard arrow keys.
List Item Execute (double click)	Indicates the user has *executed* a business service in the business service list. Fires when the user double-clicks a business service in the business service list, or when the user presses **Enter** when a business service is already selected (highlighted).

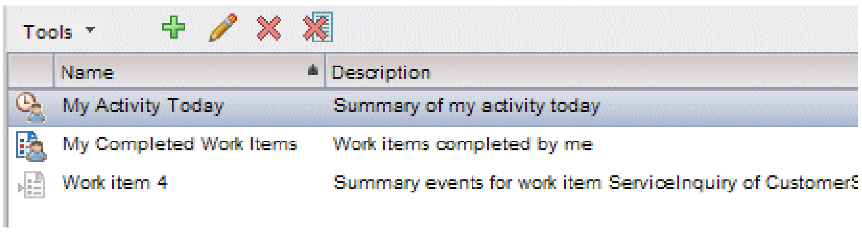
Event Views

The **Event Views** component displays a list of the event views that currently exist.

The **My Activity Today** event view is provided by default. It lists all events related to activities the currently logged-in user has performed today.

The Event Views component also contains a wizard that the user can use to create new event views.

An example event view list is shown below:



Name	Description
My Activity Today	Summary of my activity today
My Completed Work Items	Work items completed by me
Work item 4	Summary events for work item ServiceInquiry of CustomerS

For information about the functions available from this component, see the "Using Views" topic in the *TIBCO Workspace User's Guide.*

Access to each of the functions available on the event view list is controlled by properties on the **Event Views** component (in conjunction with user access sets, which are described in the **`TIBCO Workspace Customization and Configuration`** guide). For information about setting these properties, see Event Views Component Properties.

The **Event Views** component also publishes events for actions executed from the event view list. For information about these events, see Event Views Component Events.

Subscribe To:

To display the **Event Views** component, subscribe to:

- **Login** component:
  - “Login Complete” event
- **Work Items** component:
  - "Show Work Item Events" event
- **Process Instances** component:
  - "Show Instance Events" event
- **Organization Browser** component:
  - "Show Events" event
- **Organization Resource List** component:
  - "Show Events" event

Subscribing to the "Show ... Events" events cause the event view list to be refreshed so that the temporary view created by the function that fired the event is shown in the event view list.

Event Views Component Properties

The **Event Views** component contains the following properties, which are used to control access to each of the functions available from the event view list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
New Event View	Enables/disables the ability to create a new event view. Controls access to the **Create New View** button on the toolbar, as well as the **New** selection on the event view list **Tools** menu.
Edit Event View	Enables/disables the ability to edit an existing event view. Controls access to the **Edit View** button on the toolbar, as well as the **Edit** selection on the event view list **Tools** menu.
Remove Event View	Enables/disables the ability to remove an existing event view. Controls access to the **Remove View** button on the toolbar, as well as the **Remove** selection on the event view list **Tools** menu.
Open Event Viewer	Enables/disables the ability to view events. If access is denied, the event viewer is not displayed when an event view is selected in the event view list.

Event Views Component Events

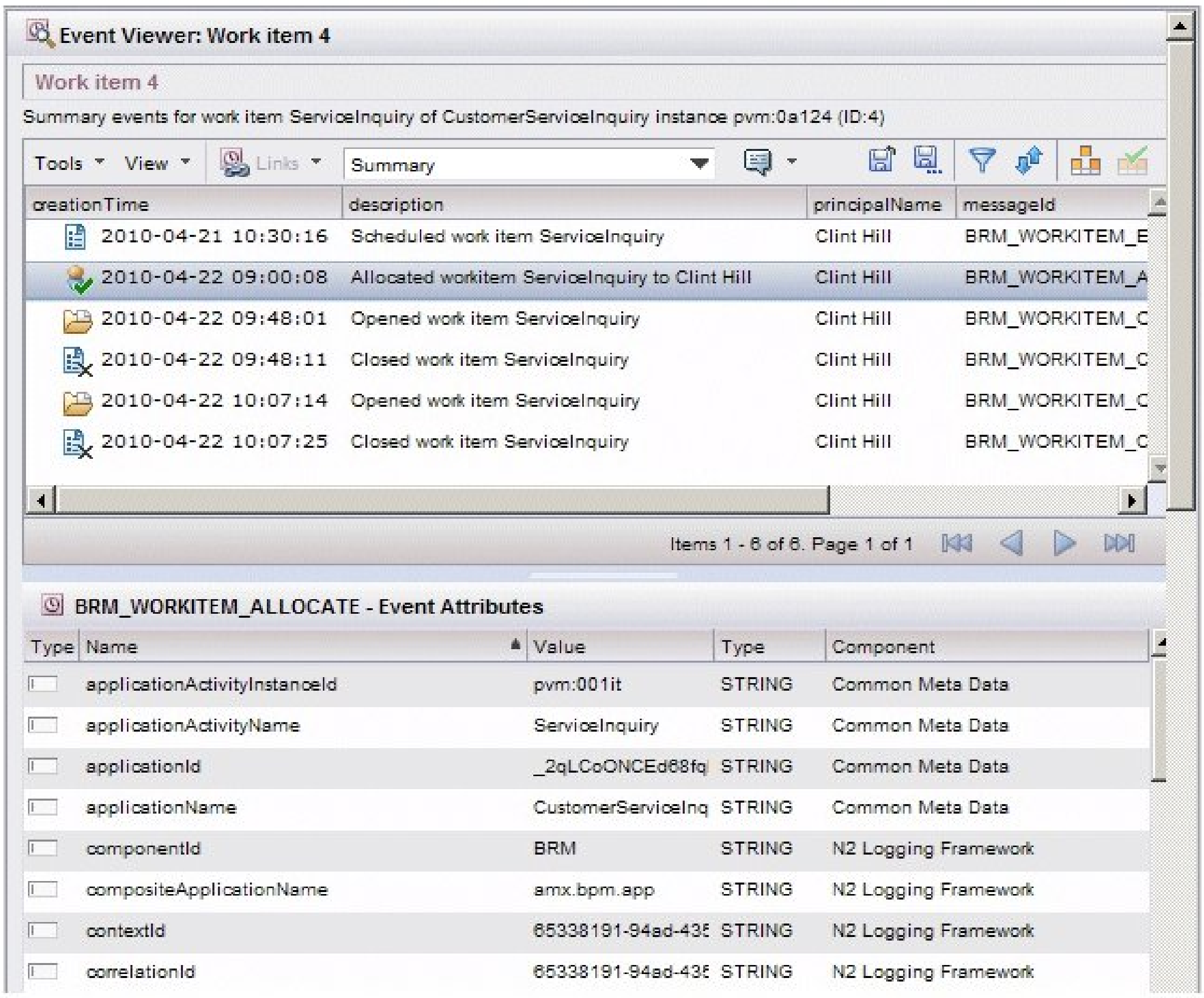
The **Event Views** component publishes the following events, which fire when the action described by the event occurs:

Event	Description
List Item Select (single click)	Indicates the user has *selected* a view in the event view list. Fires when the user single-clicks a view in the event view list, or when the user moves the highlight bar on the event view list using the keyboard arrow keys. Note, however, that this event fires only when a view other than the one currently selected is selected. In other words, clicking on the view already selected does not fire this event. This event also automatically fires when the event view list is displayed. It fires because the first view in the list is automatically selected when the list is displayed. (If there are no views in the list when it is initially displayed, the event does not fire.)
List Item Removed	Indicates the user has removed an event view. Fires when the user either clicks on the **Remove View** button or chooses the **Remove** menu selection on the event view list, then clicks **OK** on the confirmation dialog. Note that when an event view is removed, if at least one other view exists, the first view in the list is automatically selected, causing the "List Item Select (single-click)" event to also fire.
List Item Execute (double click)	Indicates the user has *executed* an event view in the event view list. Fires when the user double-clicks a view in the event view list, or when the user presses **Enter** when a view is already selected (highlighted).

Event Viewer

The **Event Viewer** component displays a list of the events in the currently selected event view.

An example event list in the Event Viewer is shown below:



Event Viewer: Work item 4

Work item 4

Summary events for work item ServiceInquiry of CustomerServiceInquiry instance pvm:0a124 (ID:4)

creationTime	description	principalName	messageId
2010-04-21 10:30:16	Scheduled work item ServiceInquiry	Clint Hill	BRM_WORKITEM_E
2010-04-22 09:00:08	Allocated workitem ServiceInquiry to Clint Hill	Clint Hill	BRM_WORKITEM_A
2010-04-22 09:48:01	Opened work item ServiceInquiry	Clint Hill	BRM_WORKITEM_C
2010-04-22 09:48:11	Closed work item ServiceInquiry	Clint Hill	BRM_WORKITEM_C
2010-04-22 10:07:14	Opened work item ServiceInquiry	Clint Hill	BRM_WORKITEM_C
2010-04-22 10:07:25	Closed work item ServiceInquiry	Clint Hill	BRM_WORKITEM_C

Items 1 - 6 of 6. Page 1 of 1

BRM_WORKITEM_ALLOCATE - Event Attributes

Type	Name	Value	Type	Component
	applicationActivityInstanceId	pvm:001it	STRING	Common Meta Data
	applicationActivityName	ServiceInquiry	STRING	Common Meta Data
	applicationId	_2qLCoONCEd68fq	STRING	Common Meta Data
	applicationName	CustomerServiceInq	STRING	Common Meta Data
	componentId	BRM	STRING	N2 Logging Framework
	compositeApplicationName	amx.bpm.app	STRING	N2 Logging Framework
	contextId	65338191-94ad-435	STRING	N2 Logging Framework
	correlationId	65338191-94ad-435	STRING	N2 Logging Framework

You can only select a single event at one time in the event viewer.

Selecting an event causes the attributes associated with that event to be displayed in the preview pane.

For information about the functions available from this component, see the "Working With Events" topic in the *TIBCO Workspace User's Guide.*

Access to each of the functions available in the event viewer is controlled by properties on the **Event Viewer** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Event Viewer Component Properties.

The **Event Viewer** component also publishes events for actions executed from the event viewer. For information about these events, see Event Viewer Component Events.

Subscribe To:

To display the **Event Viewer** component, subscribe to one of the following events:

- **EventViews** component:
  - List Item Select (single click) event
  - List Item Execute (double click) event

Event Viewer Component Properties

The **Event Viewer** component contains the following properties, which are used to control access to each of the functions available from the event list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
SaveView	Enables/disables access to the **Save View** button and menu selection on the Event Viewer.
SaveViewAs	Enables/disables access to the **Save View As** button and menu selection in the Event Viewer.
CorrelatedEvents	Enables/disables access to correlated events (i.e., being able to expand an event to view correlated events) in the Event Viewer.
EventAttributes	Enables/disables the ability to view the event attribute list in the Event Viewer.
EventLinks	Enables/disables the ability to use event links in the Event Viewer. Note that denying access prevents the **Event Links** column from being displayed, although the user can add the column back into the event list (if they have permission to do that). However, if the user does this, the **Links** drop-down menu will not provide any selections.
Filter	Enables/disables access to the **Filter** button and menu selection in the Event Viewer.
Sort	Enables/disables access to the **Sort** button and menu selection in the Event Viewer.
SelectAttributes	Enables/disables access to the **Select Attributes** button and menu selection in the Event Viewer.
SelectColumns	Enables/disables access to the **Select Columns** button and menu selection in the Event Viewer.

Event Viewer Component Events

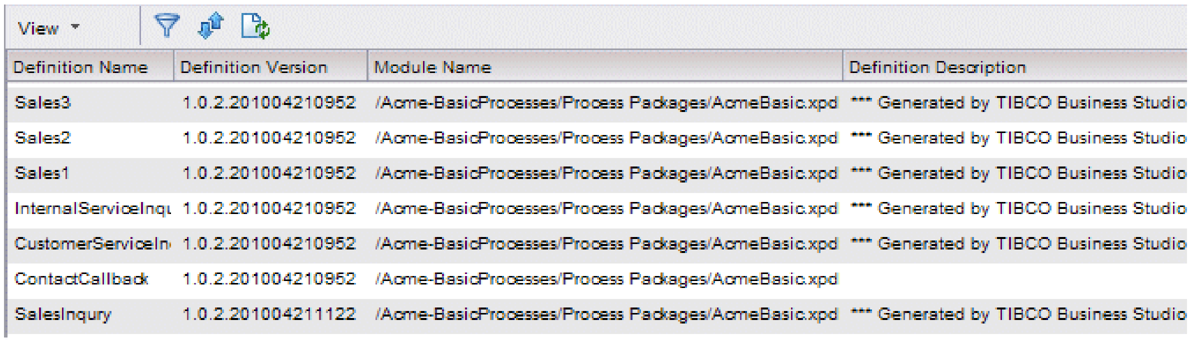
The **Event Viewer** component publishes the following events, which fire when the action described by the event occurs.

Event	Description
EventSelected	Indicates the user has *selected* an event in the Event Viewer. Fires when the user single-clicks event in the Event Viewer, or when the user moves the highlight bar on the Event Viewer using the keyboard arrow keys.
AttributeSelected	Indicates the user has selected an attribute in the Event Viewer. Fires when the user clicks on an attribute row in the list of attributes in the Event Viewer.
Save Current Event View	Indicates the user is saving the current temporary event view. Fires when the user clicks the **Save View** button on the event list toolbar, or chooses the **Save View** selection on the event list **View** menu.
Save Current Event View As	Indicates the user is saving the current event view using a different name. Fires when the user clicks the **Save View As** button on the event list toolbar, or chooses the **Save View As** selection on the event list **View** menu.

Process Templates

The **Process Templates** component displays a list of all process templates available in the system.

An example is shown below:



Definition Name	Definition Version	Module Name	Definition Description
Sales3	1.0.2.201004210952	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	*** Generated by TIBCO Business Studio
Sales2	1.0.2.201004210952	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	*** Generated by TIBCO Business Studio
Sales1	1.0.2.201004210952	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	*** Generated by TIBCO Business Studio
InternalServiceInqu	1.0.2.201004210952	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	*** Generated by TIBCO Business Studio
CustomerServiceIn	1.0.2.201004210952	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	*** Generated by TIBCO Business Studio
ContactCallback	1.0.2.201004210952	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	
SalesInqury	1.0.2.201004211122	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpd	*** Generated by TIBCO Business Studio

This component provides a flat list of available process templates that is filterable and sortable.

Also see the Process Templates Ex component, which also displays a list of the available process templates, except they are displayed in a tree structure that allows the user to select them by checking boxes.

For information about the functions available from this component, see the "Starting Process Instances" topic in the *TIBCO Workspace User's Guide.*

Access to each of the functions available in the process template list is controlled by properties on the **Process Templates** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Process Templates Component Properties.

The **Process Templates** component also publishes events for actions executed from the process template list. For information about these events, see Process Templates Component Events.

Subscribe To:

To display the **Process Templates** component, subscribe to:

- **Login** component:
  - "Login Complete" event

Process Templates Component Properties

The **Process Templates** component contains the following properties, which are used to control access to each of the functions available from the process template list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
Select Columns	Enables/disables the ability for the user to specify which columns to display in the process template list. Controls access to the **Select Columns** selection on the process template list **View** menu.
Filter	Enables/disables the ability to filter the process templates in the process template list. Controls access to the **Filter** function on the process template list.
Sort	Enables/disables the ability to sort the process templates in the process template list. Controls access to the **Sort** function on the process template list.

Process Templates Component Events

The **Process Templates** component publishes the following events, which fire when the action described by the event occurs:

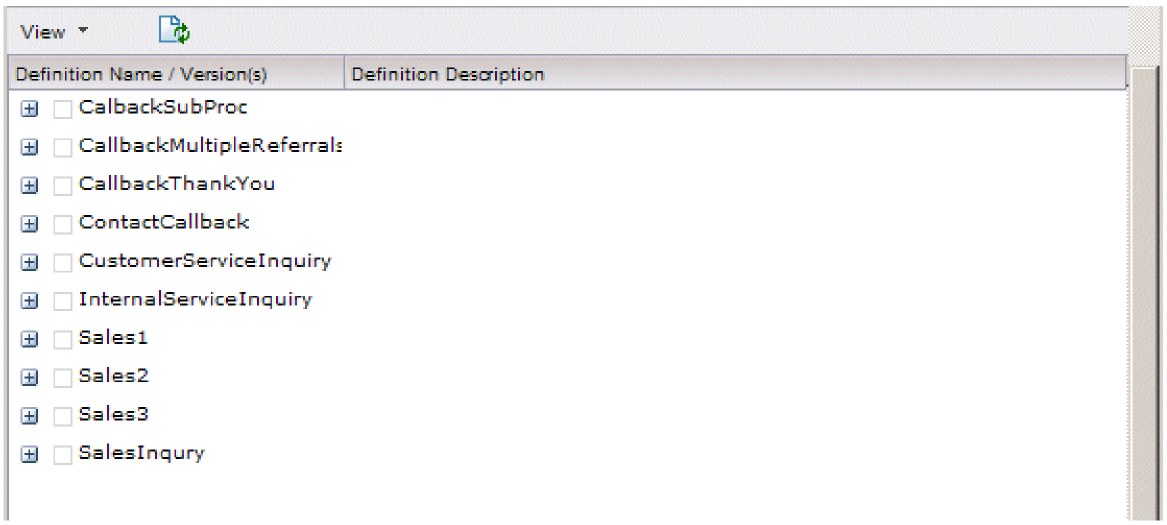
Event	Description
List Item Select (single click)	Indicates the user has *selected* a process template in the process template list. Fires when the user single-clicks a process template in the process template list, or when the user moves the highlight bar on the process template list using the keyboard arrow keys. This event also fires once each time the user "multi" selects process templates, i.e., clicking on a process template while holding down the <Ctrl> or <Shift> key. This includes "de-selecting" a process template — that is, this event fires when the user is holding down the <Ctrl> key and clicks an already selected process template to de-select it.

Event	Description
List Item Execute (double click)	Indicates the user has *executed* a process template in the process template list. Fires when the user double-clicks a process template in the process template list, or when the user presses **Enter** when a process template is already selected (highlighted).

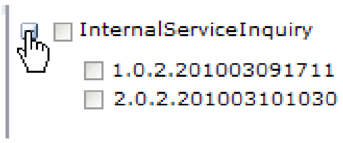
Process Templates Ex

The **Process Templates Ex** component displays a list of all process templates available in the system.

An example is shown below:



This component provides a tree structure with expandable / collapsible nodes, each node representing a process template. Expanding a node causes all versions of the process template to be displayed below the process template name. For example:



Process templates are selected by clicking in the desired boxes:

- Clicking subordinate boxes selects only those specific versions.
- Clicking a top-level box (i.e., next to the template name) causes all subordinate templates to also be selected. This causes all *future* versions of the template to be included in the selection when they are added to the system.

Also note that if the user refreshes the list of process templates, any previously selected templates are unselected. However, all of the templates/versions that were selected when the user clicked **Refresh** are included in the refresh event message.

Also see the Process Templates component, which displays a flat list of available process templates that are filterable and sortable.

For information about the functions available from this component, see the "Using Views" topic in the *TIBCO Workspace User's Guide.*

Access to each of the functions available in the process template list is controlled by properties on the **Process Templates Ex** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Process Templates Ex Component Properties.

The **Process Templates Ex** component also publishes events for actions executed from the process template list. For information about these events, see Process Templates Ex Component Events.

Subscribe To:

To display the **Process Templates Ex** component, subscribe to:

- **Login** component:
  - "Login Complete" event

Process Templates Ex Component Properties

The **Process Templates Ex** component contains the following properties, which are used to control access to each of the functions available from the process template list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
Select Columns	Enables/disables the ability for the user to specify which columns to display in the process template list. Controls access to the **Select Columns** selection on the process template list **View** menu.

Process Templates Ex Component Events

The **Process Templates Ex** component publishes the following events, which fire when the action described by the event occurs:

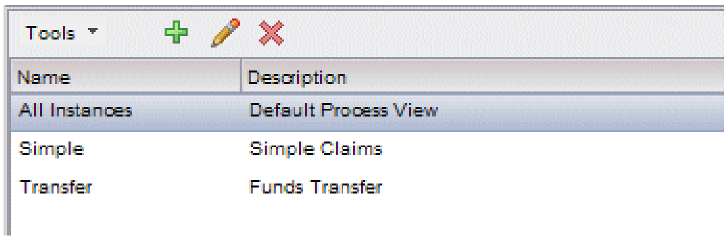
Event	Description
List Item Select (single click)	Indicates the user has *selected* a process template in the process template list. Fires when the user single-clicks a process template in the process template list, which causes the box for the selected template/version to become checked. The template/version that is selected is included in the event message.
List Refresh	Indicates the user has refreshed the process template list. Fires when the user clicks the **Refresh** button on the process template list, or selects **Refresh Process Templates** from the **View** menu. All templates/versions that are selected (checked) when a refresh is performed are included in the event message. They are automatically unchecked in the UI after the refresh.

Process Views

The **Process Views** component displays a list of the process views that the user has created. By default, one process view is provided when you first start the Workspace for the first time — the **All Instances** view.

The process view list also contains a wizard that the user can use to create new process views.

An example process view list is shown below:



Name	Description
All Instances	Default Process View
Simple	Simple Claims
Transfer	Funds Transfer

For information about the functions available from this component, see the "Using Views" topic in the *TIBCO Workspace User's Guide.*

Access to each of the functions available in the process view list is controlled by properties on the **Process Views** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Process Views Component Properties.

The **Process Views** component also publishes events for actions executed from the process view list. For information about these events, see Process Views Component Events.

Subscribe To:

To display the **Process Views** component, subscribe to:

- **Login** component:
  – "Login Complete" event

Process Views Component Properties

The **Process Views** component contains the following properties, which are used to control access to each of the functions available from the process view list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
New Process View	Enables/disables the ability to create a new process view. Controls access to the **Create New View** button on the toolbar, as well as the **New** selection on the process view list **Tools** menu.
Edit Process View	Enables/disables the ability to edit an existing process view. Controls access to the **Edit View** button on the toolbar, as well as the **Edit** selection on the process view list **Tools** menu.

Property	Description
Remove Process View	Enables/disables the ability to remove an existing process view. Controls access to the **Remove View** button on the toolbar, as well as the **Remove** selection on the process view list **Tools** menu.

Process Views Component Events

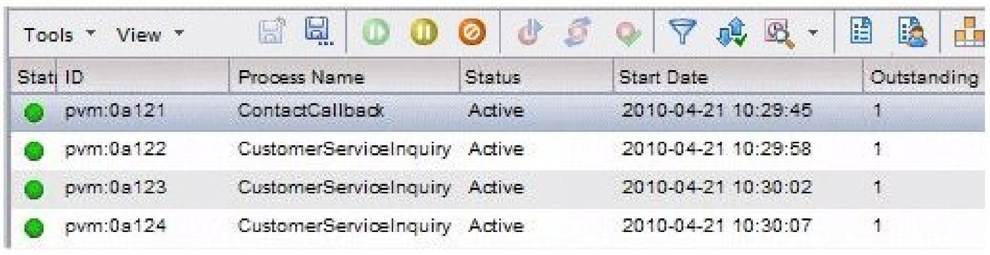
The **Process Views** component publishes the following events, which fire when the action described by the event occurs:

Event	Description
List Item Select (single click)	Indicates the user has *selected* a view in the process view list. Fires when the user single-clicks a view in the process view list, or when the user moves the highlight bar on the process view list using the keyboard arrow keys. Note, however, that this event fires only when a view other than the one currently selected is selected. In other words, clicking on the view already selected does not fire this event. This event also automatically fires when the process view list is displayed. It fires because the first view in the list is automatically selected when the list is displayed. (If there are no views in the list when it is initially displayed, the event does not fire.)
List Item Removed	Indicates the user has removed a process view. Fires when the user either clicks on the **Remove View** button or chooses the **Remove** menu selection on the process view list, then clicks **OK** on the confirmation dialog. Note that when a process view is removed, if at least one other view exists, the first view in the list is automatically selected, causing the "List Item Select (single-click)" event to also fire.
List Item Execute (double click)	Indicates the user has *executed* a process view in the process view list. Fires when the user double-clicks a view in the process view list, or when the user presses **Enter** when a view is already selected (highlighted).

Process Instances

The **Process Instances** component displays a list of all of the process instances in a specific process instance view.

An example process instance list is illustrated below:



Stati	ID	Process Name	Status	Start Date	Outstanding
●	pvm:0a121	ContactCallback	Active	2010-04-21 10:29:45	1
●	pvm:0a122	CustomerServiceInquiry	Active	2010-04-21 10:29:58	1
●	pvm:0a123	CustomerServiceInquiry	Active	2010-04-21 10:30:02	1
●	pvm:0a124	CustomerServiceInquiry	Active	2010-04-21 10:30:07	1

For details about the functions available from the process instance list, see the "Working With Process Instances" topic in the *TIBCO Workspace User's Guide*.

Access to each of the functions available in the process instance list is controlled by properties on the **Process Instances** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Process Instances Component Properties.

The **Process Instances** component also publishes events for actions executed from the process instance list. For information about these events, see Process Instances Component Events.

Subscribe To:

To display the **Process Instances** component, subscribe to one of the following events:

- **Process Views** component:
  - List Item Select (single click) event
  - List Item Execute (double click) event

Process Instances Component Properties

The **Process Instances** component contains the following properties, which are used to control access to each of the functions available from the process instance list (these properties are used in conjunction with "user access sets" to control access — for more information, see The Properties Editor is used to specify property definitions for Workspace components.).

Property	Description
Define Halted View	Enables/disables the ability to create a halted process instance view. Controls access to the **Halted instance view** selection on the **New Process View** and Edit Process View dialogs.
Save View	Enables/disables the ability to save the temporary process view. Controls access to the **Save View** button on the toolbar, as well as the **Save View** selection on the process instance list **View** menu.
Save View As	Enables/disables the ability to save the temporary process view using a different name. Controls access to the **Save View As** button on the toolbar, as well as the **Save View As** selection on the process instance list **View** menu.
Resume Process Instance	Enables/disables the ability to resume a suspended process instance. Controls access to the **Resume** button on the toolbar, as well as the **Resume Process Instance(s)** selection on the process instance list **View** menu.

Property	Description
Cancel Process Instance	Enables/disables the ability to cancel a process instance. Controls access to the **Cancel** button on the toolbar, as well as the **Cancel Process Instance(s)** selection on the process instance list **View** menu.
Suspend Process Instance	Enables/disables the ability to suspend a process instance. Controls access to the **Suspend** button on the toolbar, as well as the **Suspend Process Instance(s)** selection on the process instance list **View** menu.
Resume Halted Process Instance	Enables/disables the ability to resume a process instance that has been halted. Controls access to the **Resume Halted Process Instance(s)** button and menu selection on the process instance list.
Retry Halted Process Instance	Enables/disables the ability to retry a process instance that has been halted. Controls access to the **Retry Halted Process Instance(s)** button and menu selection on the process instance list.
Ignore Halted Process Instance	Enables/disables the ability to ignore a process instance that has been halted. Controls access to the **Ignore Fault For Halted Process Instance(s)** button and menu selection on the process instance list.
Select Columns	Enables/disables the ability to specify which columns to display in the process instance list. Controls access to the **Select Columns** button on the toolbar, as well as the **Select Columns** selection on the process instance list **View** menu.
Filter	Enables/disables the ability to filter the process instances in the process instance list. Controls access to the **Filter** button on the toolbar, as well as the **Filter Process Instance(s)** selection on the process instance list **View** menu.
Sort	Enables/disables the ability to sort the process instances in the process instance list. Controls access to the **Sort** button on the toolbar, as well as the **Sort Process Instance(s)** selection on the process instance list **View** menu.

Property	Description
Show Outstanding	Enables/disables the ability to display list of outstanding work items for the selected process instance. Controls access to the **Show Outstanding Work Items** button on the toolbar, as well as the **Show Outstanding Work Items** selection on the process instance list **Tools** menu.
Show Supervised Admin	Enables/disables the ability to display list of supervised outstanding work items for the selected process instance. Controls access to the **Show Supervised Outstanding Work Items** button on the toolbar, as well as the **Show Supervised List of Outstanding Work Items** selection on the process instance list **Tools** menu.
Event Viewer	Enables/disables the ability to display a list of events related to the selected process instance. Controls access to the **View Events** button on the toolbar, as well as the **Open Event Viewer** selection on the process instance list **View** menu.

Process Instances Component Events

The **Process Instances** component publishes the following events, which fire when the action described by the event occurs.

Event	Description
List Item Select (single click)	Indicates the user has *selected* an instance in the process instance list. Fires when the user single-clicks an instance in the process instance list, or when the user moves the highlight bar on the process instance list using the keyboard arrow keys.
List Item Execute (double click)	Indicates the user has *executed* an instance in the process instance list. Fires when the user double-clicks an instance in the process instance list, or when the user presses **Enter** when an instance is already selected (highlighted).
Cancel Instance	Indicates the user is cancelling one or more process instances. Fires when the user clicks the **Cancel** button on the process instance list toolbar, or chooses the **Cancel Process Instance(s)** selection on the process instance list **View** menu.

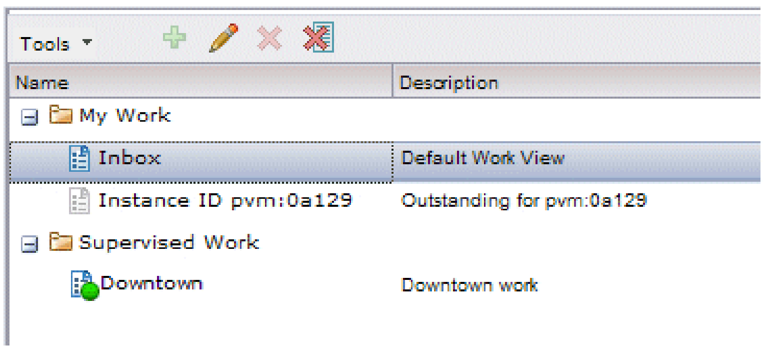
Event	Description
Suspend Instance	Indicates the user is suspending one or more process instances. Fires when the user clicks the **Suspend** button on the process instance list toolbar, or chooses the **Suspend Process Instance(s)** selection on the process instance list **View** menu.
Resume Instance	Indicates the user is resuming one or more suspended process instances. Fires when the user clicks the **Resume** button on the process instance list toolbar, or chooses the **Resume Process Instance(s)** selection on the process instance list **View** menu.
Resume Halted Instance	Indicates the user is resuming one or more halted process instances. Fires when the user clicks the **Resume Halted Process Instance(s)** button on the process instance list toolbar, or chooses the **Resume Halted Process Instance(s)** selection on the process instance list **Tools** menu.
Retry Halted Instance	Indicates the user is retrying one or more halted process instances. Fires when the user clicks the **Retry Halted Process Instance(s)** button on the process instance list toolbar, or chooses the **Retry Halted Process Instance(s)** selection on the process instance list **Tools** menu.
Ignore Halted Instance	Indicates the user is ignoring faults for one or more halted process instances. Fires when the user clicks the **Ignore Fault For Halted Process Instance(s)** button on the process instance list toolbar, or chooses the **Ignore Fault For Halted Process Instance(s)** selection on the process instance list **Tools** menu.
List Outstanding Work Items	Indicates the user is displaying a list of outstanding work items for the selected process instance. Fires when the user clicks either the **Show Outstanding Work Items** or the **Show Supervised List of Outstanding Work Items** button on the process instance list toolbar, or chooses either the **Show Outstanding Work Items** or **Show Supervised List of Outstanding Work Items** selection on the process instance list **Tools** menu.

Event	Description
Save Current Process View	Indicates the user is saving the current temporary process view. Fires when the user clicks the **Save View** button on the process instance list toolbar, or chooses the **Save View** selection on the process instance list **View** menu.
Save Current Process View As	Indicates the user is saving the current temporary process view using a different name. Fires when the user clicks the **Save View As** button on the process instance list toolbar, or chooses the **Save View As** selection on the process instance list **View** menu.
Show Instance Events	Indicates the user is creating an event view containing events related to the selected process instance. Fires when the user makes a selection from the drop-down list that is displayed when the user either clicks on the button on the process instance list toolbar, selects **Open Event Viewer** from the process instance list **Tools** menu, or right-clicks on the selected process instance and selects **Open Event Viewer**.

Work Views

The **Work Views** component displays a list of the work views that the user has created. Note that every user has an "Inbox" work view that, by default, displays a list of all work items that have been offered or allocated to the user (although the user can specify a filter for the Inbox so that it displays a subset of the work items available to the user).

An example work view list is illustrated below:



Name	Description
My Work	
Inbox	Default Work View
Instance ID pvm:0a129	Outstanding for pvm:0a129
Supervised Work	
Downtown	Downtown work

For details about the functions available from the work view list, see the "Using Views" topic in the *TIBCO Workspace User's Guide*.

Access to each of the functions available in the work view list is controlled by properties on the **Work Views** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Work Views Component Properties.

The **Work Views** component also publishes events for actions executed from the work view list. For information about these events, see Work Views Component Events.

Subscribe To:

To display the **Work Views** component, subscribe to:

- **Login** component:
  - “Login Complete” event

Work Views Component Properties

The **Work Views** components contain the following properties, which are used to control access to each of the functions available from the work view list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor.)

Property	Description
Work View List	Enables/disables the ability to view the work view list.
New Work View	Enables/disables the ability to create a new work view. Controls access to the **Create New View** button on the toolbar, as well as the **New** selection on the work view list **Tools** menu.
Edit Work View	Enables/disables the ability to edit an existing work view. Controls access to the **Edit View** button on the toolbar, as well as the **Edit** selection on the work view list **Tools** menu.
Remove Work View	Enables/disables the ability to remove an existing work view. Controls access to the **Remove View** button on the toolbar, as well as the **Remove** selection on the work view list **Tools** menu.
WorkItem	Enables/disables the ability to view personal work views.
SupervisedWorkItem	Enables/disables the ability to view supervised work views.

Work Views Component Events

The **Work Views** component publishes the following events, which fire when the action described by the event occurs:

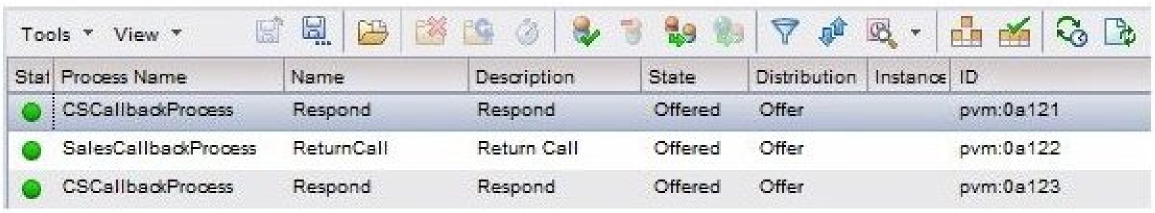
Event	Description
List Item Select (single click)	Indicates the user has *selected* any line in the work view list. This includes individual work views, as well as the "My Work" and "Supervised Work" folders. Fires when the user single-clicks a line in the work view list, or when the user moves the highlight bar on the work view list using the keyboard arrow keys. This event also automatically fires when the work view list is displayed. It fires because the Inbox work view is automatically selected when the list is displayed.

Event	Description
List Item Removed	Indicates the user has removed a work view. Fires when the user either clicks on the **Remove View** button or chooses the **Remove** menu selection on the work view list, then clicks **OK** on the confirmation dialog. Note that when a work view is removed, another work view is automatically selected, causing the "List Item Select (single-click)" event to also fire (by default, the Inbox is automatically selected, but that can be changed in Options).
List Item Execute (double click)	Indicates the user has *executed* a view in the work view list. Fires when the user double-clicks a view in the work view list, or when the user presses **Enter** when a view is already selected (highlighted).

Work Items

The **Work Items** component displays a list of all of the work items in a specific work view.

An example work item list is illustrated below:



Stat	Process Name	Name	Description	State	Distribution	Instance	ID
●	CSCallbackProcess	Respond	Respond	Offered	Offer		pvm:0a121
●	SalesCallbackProcess	ReturnCall	Return Call	Offered	Offer		pvm:0a122
●	CSCallbackProcess	Respond	Respond	Offered	Offer		pvm:0a123

For details about the functions available from the work item list, see the "Working With Work Items" topic in the *TIBCO Workspace User's Guide*.

Access to each of the functions available in the work item list is controlled by properties on the **Work Items** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Work Items Component Properties.

The **Work Views** component also publishes events for actions executed from the work view list. For information about these events, see Work Items Component Events.

Pageflow definitions can include branching that would require the simultaneous handling of multiple user tasks, each requiring the display of a form (this same scenario could also occur in embedded sub-processes set up for chained execution).

Workspace cannot currently display multiple forms resulting from these *parallel pageflows*. If one is encountered in a WCC application (including the Workspace application), a warning message is displayed. In some situations, none of the forms can be displayed and execution of the pageflow (or chaining in the sub-process) cannot proceed. If a parallel pageflow is encountered inside of a business service, it may be able to arbitrarily display one of the forms, but an exception may occur later when the form is submitted.

Subscribe To:

To display the **Work Items** component, subscribe to one of the following events:

- **Work Views** component:

- List Item Select (single click) event
- List Item Execute (double click) event

Work Items Component Properties

The **Work Items** component contains the following properties, which are used to control access to each of the functions available from the work item list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Note that the properties on the work item list are divided between "Supervised Work Items" and "Work Items":

- "Supervised Work Items" represent the functions available on the work item list when a supervised work view is currently selected (i.e., a work view under the **Supervised Work** folder in the work view list).
- "Work Items" represent the functions available on the work item list when a personal work view is currently selected (i.e., a work view under the **My Work** folder in the work view list).

Property	Description
SupervisedWorkItem	Enables/disables the ability to access supervised work views. If disabled, the work item list is not displayed when a supervised work view is selected.
SupervisedWorkItem - Cancel	Enables/disables the ability to cancel work items in a supervised work item list. Controls access to the **Cancel Work Item(s)** button / menu selection when a supervised work view is selected.
SupervisedWorkItem - Skip	Enables/disables the ability to skip work items in a supervised work item list. Controls access to the **Skip Work Item(s)** button / menu selection when a supervised work view is selected.
SupervisedWorkItem - Reoffer	Enables/disables the ability to re-offer work items in a supervised work item list. Controls access to the **Re-Offer Work Item(s)** button / menu selection when a supervised work view is selected.
SupervisedWorkItem - AllocateToAnother	Enables/disables the ability to allocate work items to another user from a supervised work item list. Controls access to both the **Allocate Work Item(s) To Offer Set** and the **Allocate Work Item(s) To World** functions when a supervised work view is selected.

Property	Description
SupervisedWorkItem - AllocateToAnother - CanAllocateFromOfferSet	Enables/disables the ability to allocate work items to another user from the original offer set. Controls access to the **Allocate Work Item(s) To Offer Set** function when a supervised work view is selected.
SupervisedWorkItem - AllocateToAnother - CanAllocateFromOrganization	Enables/disables the ability to allocate work items to another user from all available resources. Controls access to the **Allocate Work Item(s) To World** function when a supervised work view is selected.
SupervisedWorkItem - AllocateToAnother - ShowResourcePreview	Enables/disables the ability to view resource details on the Allocate To... dialogs. Controls access to the **Toggle Preview** button and menu selection on both the **Allocate Work Item(s) To Offer Set** and the Allocate Work Item(s) To World dialogs when a supervised work view is selected.
SupervisedWorkItem - Event Viewer	Enables/disables the ability to create an event view containing the events related to the selected work item. Controls access to the **Open Event Viewer** button / menu selection when a supervised work view is selected.
SupervisedWorkItem - Filter	Enables/disables the ability to filter the work items in a supervised work item list. Controls access to the **Filter** button / menu selection when a supervised work view is selected.
SupervisedWorkItem - Sort	Enables/disables the ability to sort the work items in a supervised work item list. Controls access to the **Sorting** button / menu selection when a supervised work view is selected.
SupervisedWorkItem - Auto-Refresh	Enables/disables the ability to specify that the supervised work item list be automatically refreshed at a given time interval. Controls access to the **Auto-Refresh** button / menu selection when a supervised work view is selected.

Property	Description
SupervisedWorkItem - Select Columns	Enables/disables the ability to specify which columns to display in the supervised work item list. Controls access to the **Select Columns** button on the toolbar, as well as the **Select Columns** selection on the **View** menu when a supervised work view is selected.
WorkItem	Enables/disables the ability to access personal work views. If disabled, the work item list is not displayed when a personal work view is selected.
WorkItem - Save View	Enables/disables the ability to save the temporary work view. Controls access to the **Save View** button / menu selection when a personal work view is selected.
WorkItem - Save View As	Enables/disables the ability to save the temporary work view using a different name. Controls access to the **Save View As** button / menu selection when a personal work view is selected.
WorkItem - Open	Enables/disables the ability to open (lock) a work item. Controls access to the **Open Selected Work Item(s)** button / menu selection when a personal work view is selected.
WorkItem - Open-Next	Enables/disables the ability to open (lock) the next work item in the list. Controls access to the **Open Next Work Item** button / menu selection when a personal work view is selected.
WorkItem - Auto-Repeat	Enables/disables the ability to automatically open (lock) work items. Controls access to the **Auto-Repeat Open Work Item** button / menu selection when a personal work view is selected.
WorkItem - Cancel	Enables/disables the ability to unlock (cancel) a work item that had been locked by another user. Controls access to the **Cancel Work Item(s)** button / menu selection when a personal work view is selected.

Property	Description
WorkItem - Skip	Enables/disables the ability to skip work items. Controls access to the **Skip Work Item(s)** button / menu selection when a personal work view is selected.
WorkItem - Pend	Enables/disables the ability to pend work items. Controls access to the **Pend Work Item(s)** button / menu selection when a personal work view is selected.
WorkItem - AllocateToSelf	Enables/disables the ability to allocate a work item to yourself, removing it from other user's work lists. Controls access to the **Allocate Work Item(s) to Self** button / menu selection when a personal work view is selected.
WorkItem - Reoffer	Enables/disables the ability to change work items that are allocated to you, back to an Offered state. They are offered to the users to whom they were originally offered. Controls access to the **Re-Offer Work Item(s)** button / menu selection when a personal work view is selected.
WorkItem - AllocateToAnother	Enables/disables the ability to allocate work items to another user from a personal work item list. Controls access to both **Allocate Work Item(s) To Offer Set** and the **Allocate Work Item(s) To World** functions when a personal work view is selected.
WorkItem - AllocateToAnother - CanAllocateFromOfferSet	Enables/disables the ability to allocate work items to another user from the original offer set. Controls access to the **Allocate Work Item(s) To Offer Set** function when a personal work view is selected.
WorkItem - AllocateToAnother - CanAllocateFromOrganization	Enables/disables the ability to allocate work items to another user from all available resources. Controls access to the **Allocate Work Item(s) To World** function when a personal work view is selected.(1)

Property	Description
WorkItem - AllocateToAnother - ShowResourcePreview	Enables/disables the ability to view resource details on the "Allocate" dialogs. Controls access to the **Toggle Preview** button and menu selection on both the **Allocate Work Item(s) To Offer Set** and the Allocate Work Item(s) To World dialogs when a personal work view is selected.
WorkItem - Event Viewer	Enables/disables the ability to create an event view containing the events related to the selected work item. Controls access to the **Open Event Viewer** button / menu selection when a personal work view is selected.
WorkItem - Filter	Enables/disables the ability to filter the work items in the work item list. Controls access to the **Filter** button / menu selection when a personal work view is selected.
WorkItem - Sort	Enables/disables the ability to sort the work items in the work item list. Controls access to the **Sorting** button / menu selection when a personal work view is selected.
WorkItem - Auto-Refresh	Enables/disables the ability to specify that the work item list be automatically refreshed at a given time interval. Controls access to the **Auto-Refresh** button / menu selection when a personal work view is selected.
WorkItem - Select Columns	Enables/disables the ability to specify which columns to display in the work item list. Controls access to the **Select Columns** button on the toolbar, as well as the **Select Columns** selection on the **View** menu when a personal work view is selected.

(1) You can also control access to parts of the Allocate Work Item(s) To World dialog by using the properties on the Organization Browser and Organization Resource List components, as those components are used to present the organization model and resource list in the Allocate Work Item(s) To World dialog.

Work Items Component Events

The **Work Items** component publishes the following events, which fire when the action described by the event occurs.

Event	Description
List Item Select (single click)	Indicates the user has *selected* a work item in the work item list. Fires when the user single-clicks a work item in the work item list, or when the user moves the highlight bar on the work item list using the keyboard arrow keys.
List Item Execute (double click)	Indicates the user has *executed* a work item in the work item list. Fires when the user double-clicks a work item in the work item list, or when the user presses **Enter** when a work item is already selected (highlighted).
Open Work Item(s)	Indicates the user has opened a work item. Fires when the user clicks the **Open Selected Work Item(s)** button on the work item list toolbar, or chooses the **Open Selected Work Item(s)** menu selection on the work item list **Tools** menu.
Open Next Work Item	Indicates the user is opening the next available work item in the work item list. Fires when the user clicks the **Open Next Work Item** button on the work item list toolbar, or chooses the **Open Next Work Item** menu selection on the work item list **Tools** menu.
Cancel Work Item(s)	Indicates the user is cancelling a work item that had been opened by another user. Fires when the user clicks the **Cancel Work Item(s)** button on the work item list toolbar, or chooses the **Cancel Work Item(s)** menu selection on the work item list **Tools** menu.
Skip Work Item(s)	Indicates the user is skipping a work item. Fires when the user clicks the **Skip Work Item(s)** button on the work item list toolbar, or chooses the **Skip Work Item(s)** menu selection on the work item list **Tools** menu.
Pend Work Item(s)	Indicates the user is pending a work item. Fires when the user clicks the **Pend Work Item(s)** button on the work item list toolbar, or chooses the **Pend Work Item(s)** menu selection on the work item list **Tools** menu.
Allocate Work Item(s) to Self	Indicates the user is allocating one or more work items to himself. Fires when the user clicks the **Allocate Work Item(s) to Self** button on the work item list toolbar, or chooses the **Allocate Work Item(s) to Self** menu selection on the work item list **Tools** menu.

Event	Description
Re-Offer Work Item(s)	Indicates the user is re-offering work items to the users to whom they were originally offered. Fires when the user clicks the **Re-Offer Work Item(s)** button on the work item list toolbar, or chooses the **Re-Offer Work Item(s)** menu selection on the work item list **Tools** menu.
Allocate Work Item(s) to Offer Set	Indicates the user is allocating one or more work items to one or more users who were in the set of users to whom the work items were originally offered. Fires when the user clicks the **Allocate Work Item(s) to Offer Set** button / menu selection on the work item list.
Allocate Work Item(s) to World	Indicates the user is allocating one or more work items to another user they are choosing from an organization model entity. Fires when the user clicks the **Allocate Work Item(s) to World** button / menu selection on the work item list.
Show Work Item Events	Indicates the user is creating a temporary event view containing events related to the selected work item. Fires when the user makes a selection from the drop-down list that is displayed when the user either clicks on the button on the work item list toolbar, selects **Open Event Viewer** from the work item list **Tools** menu, or right-clicks on the selected work item and selects **Open Event Viewer**.
Save Current Work View	Indicates the user is saving the current temporary work view. Fires when the user clicks the **Save View** button on the work item list toolbar, or chooses the **Save View** selection on the work item list **View** menu.
Save Current Work View As	Indicates the user is saving the current temporary work view using a different name. Fires when the user clicks the **Save View As** button on the work item list toolbar, or chooses the **Save View As** selection on the work item list **View** menu.

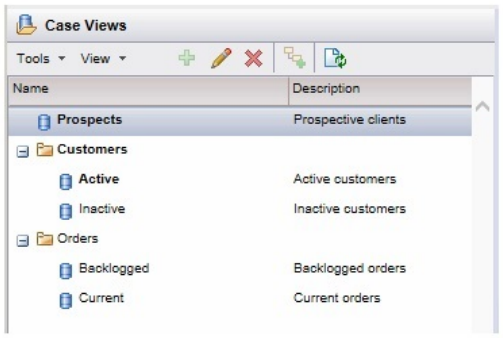
Data Views

The **Data Views** component displays a list of the data views that currently exist.

As of version 4.0 of Workspace, "data views" are now called "case views". However, the term "data view" is still used in General Interface Builder; the **Data Views** component appears as "Case Views" in the application.

The **Data Views** component also contains a wizard that the user can use to create new data views.

An example data views list is shown below:



Name	Description
Prospects	Prospective clients
Customers	
Active	Active customers
Inactive	Inactive customers
Orders	
Backlogged	Backlogged orders
Current	Current orders

For information about the functions available from this component, see the "Using Views" topic in the *TIBCO Workspace User's Guide.*

Access to each of the functions available on the data views list is controlled by properties on the **Data Views** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Data Views Component Properties.

The **Data Views** component also publishes events for actions executed from the data views list. For information about these events, see Data Views Component Events.

Subscribe To:

To display the **Data Views** component, subscribe to:

- **Login** component:
  - "Login Complete" event

Data Views Component Properties

The **Data Views** component contains the following properties, which are used to control access to each of the functions available from the data views list (these properties are used in conjunction with "user access sets" to control access — for more information, see Properties Editor).

Property	Description
New Data View	Enables/disables the ability to create a new data view. Controls access to the **Create New View** button on the toolbar, as well as the **New** selection on the data views list **Tools** menu.
Edit Data View	Enables/disables the ability to edit an existing data view. Controls access to the **Edit View** button on the toolbar, as well as the **Edit** selection on the data views list **Tools** menu.
Remove Data View	Enables/disables the ability to remove an existing data view. Controls access to the **Remove View** button on the toolbar, as well as the **Remove** selection on the data views list **Tools** menu.
New Data View Category	Enables/disables the ability to create a new data view category. Controls access to the **Add Category** button on the toolbar, as well as the **Add Category** selection on the data views list **Tools** menu.

Data Views Component Events

The Data **Views** component publishes the following events, which fire when the action described by the event occurs:

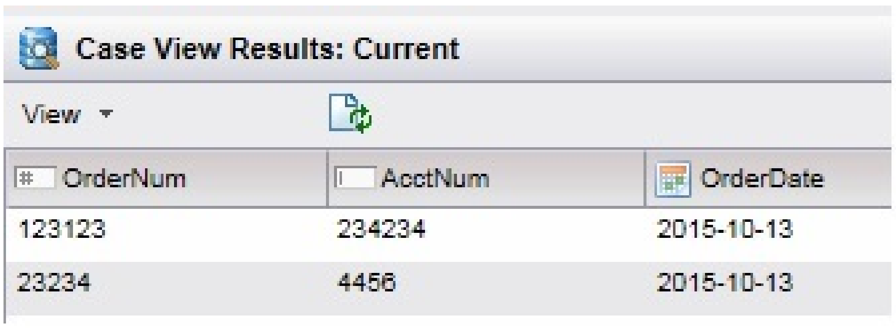
Event	Description
List Item Select (single click)	Indicates the user has *selected* a view in the data views list. Fires when the user single-clicks a view in the data views list, or when the user moves the highlight bar on the data views list using the keyboard arrow keys. Note, however, that this event fires only when a view other than the one currently selected is selected. In other words, clicking on the view already selected does not fire this event. This event also automatically fires when the data views list is displayed. It fires because the first view in the list is automatically selected when the list is displayed. (If there are no views in the list when it is initially displayed, the event does not fire.)
List Item Removed	Indicates the user has removed a data view. Fires when the user either clicks on the **Remove View** button or chooses the **Remove** menu selection on the data views list, then clicks **OK** on the confirmation dialog.
List Item Execute (double click)	Indicates the user has *executed* a data view in the data views list. Fires when the user double-clicks a view in the data views list, or when the user presses **Enter** when a view is already selected (highlighted).

Data View List

The **Data View List** component lists the cases for the selected case view.

As of version 4.0 of Workspace, "data views" are now called "case views". However, the term "data view" is still used in General Interface Builder; the **Data View List** component appears as the "Case View Results" in the application.

An example data view list is shown below:



OrderNum	AcctNum	OrderDate
123123	234234	2015-10-13
23234	4456	2015-10-13

For information about the functions available from this component, see the "Using Views" topic in the *TIBCO Workspace User's Guide.*

The **Data View List** component does not contain any properties.

The **Data View List** component publishes events for actions executed from the case reference list. For information about these events, see Data View List Component Events.

Subscribe To:

To display the **Data View List** component, subscribe to:

- **Data Views** component:
  - List Item Select (single click) event
  - List Item Execute (double click) event

Data View List Component Events

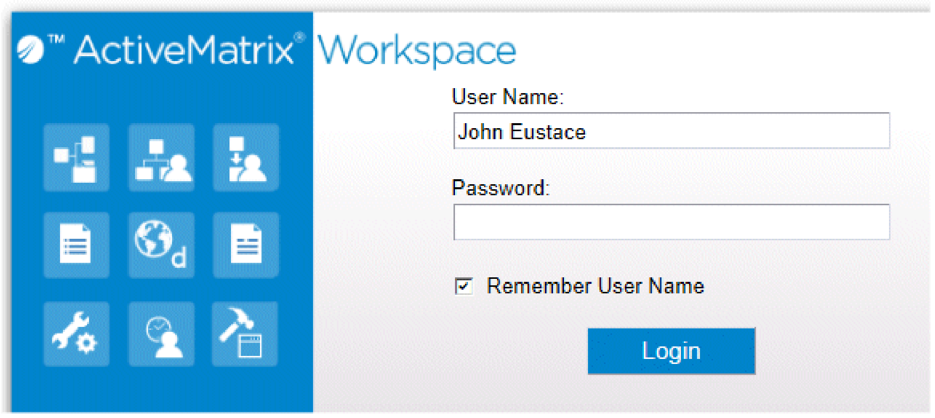
The **Data View List** component publishes the following events, which fire when the action described by the event occurs:

Event	Description
List Item Select (single click)	Indicates the user has *selected* a case in the data views list. Fires when the user single-clicks a case in the data views list, or when the user moves the highlight bar on the data views list using the keyboard arrow keys. Note, however, that this event fires only when a case other than the one currently selected is selected. In other words, clicking on the case already selected does not fire this event.
List Item Execute (double click)	Indicates the user has *executed* a data view in the data views list. Fires when the user double-clicks a case in the data views list, or when the user presses **Enter** when a case is already selected (highlighted).

Login Component

The **Login** component provides an interface for logging into the application.

This component displays the Login dialog, which accepts the user's credentials and submits them to the Action Processor / node for authentication.



User Name: John Eustace

☑ Remember User Name

The Login dialog contains the following fields:

- **User Name** - A user ID (name) for each user is established when they are created using the Organization Browser. For information, see the *TIBCO Organization Browser User's Guide*.
- **Password** - User passwords are specified in the LDAP source from which the user was created by the Organization Browser.

For more information about logging in, see the "Accessing Workspace" topic in the *TIBCO Workspace User's Guide*.

For information about logging in programmatically using a Tools Interface method, see login.

For information about utilizing a login from a previous login session to login to a WCC application running in an Iframe, see Using WCC Components in Mash-Ups.

Subscribe To:

To display the **Login** component, subscribe to:

- None

  The **Login** component does not require that any data be passed to it to be displayed, that is, it does not need to subscribe to an event of another component.

Login Component Properties

The **Login** component contains the following property:

Property	Description
Remember	Enables / disables the **Remember User Name** check box, which allows users to persist their user name (**User Id** field).

Login Component Events

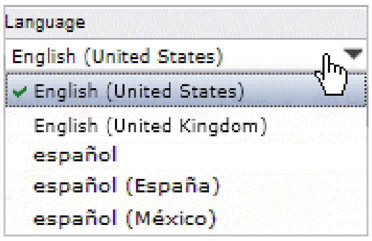
The **Login** component publishes the following event:

Event	Description
LoginComplete	Indicates a successful login. Event fires when user authentication is complete.

LocaleSelector Component

The **LocaleSelector** component allows the user to change the language in which the application is displayed.

For example:



The languages listed depend on the "language packs" that have been installed on the system, or languages that have been added manually. For more information, see the "Localization" topic in the *TIBCO Workspace Configuration and Customization* guide.

A language selection from this component takes effect immediately, and is persisted (i.e., the application will continue to be displayed in the selected language until it is changed again, either using this function, or the **Language** user option on the Options dialog).

When a language is selected from the LocaleSelector component, the **setAppLocale** Application method is called, which publishes the **localeChanged** PageBus event. All WCC components inherently subscribe to the **localeChanged** event so that they can be refreshed when the locale is changed. (Note, however, the **localeChanged** event does *not* appear in the Event Editor in General Interface Builder.) For information about **setAppLocale**, see setAppLocale.

Subscribe To:

To display the **LocaleSelector** component, subscribe to:

- **Login** component:

  "Login Complete" event

LocaleSelector Properties

None.

LocaleSelector Events

None.

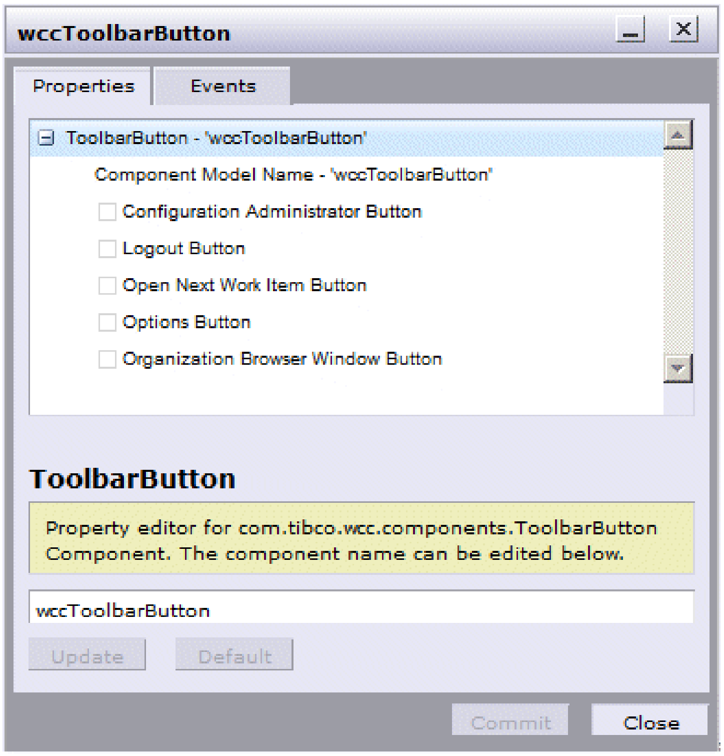
Toolbar Button Component

The **Toolbar Button** component allows you to display a number of buttons that perform various functions.

The available buttons are:

- Organization Browser Window Button - displays the Configuration Administrator dialog, which is used to configure deployed applications.
- Logout Button - logs the user out of the application.
- Open Next Work Item Button - opens the next available work item from the work item list.
- Options Button - displays the Options dialog, which is used to establish default settings for the currently logged-in user.
- Organization Browser Window Button - displays the Organization Resource Browser component, which is a composite component used to browse the organization model, create LDAP containers, map users to groups/positions, etc.

This component works a little differently than the other WCC components. The Properties Editor is used to specify which of the buttons listed above will be displayed by the Toolbar Button component.



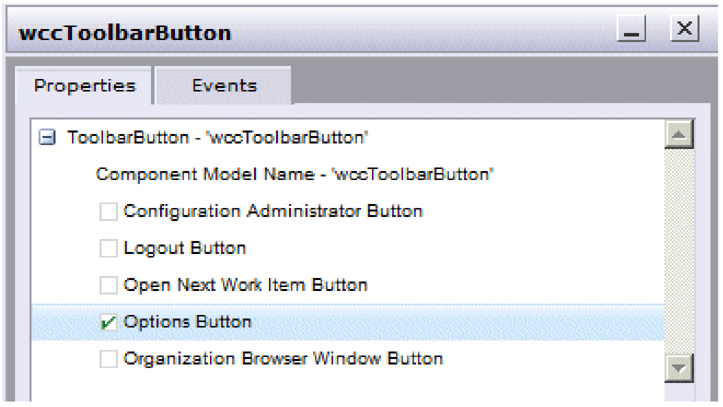
ToolbarButton - 'wccToolbarButton'
Component Model Name - 'wccToolbarButton'
Configuration Administrator Button
Logout Button
Open Next Work Item Button
Options Button
Organization Browser Window Button

ToolbarButton

Property editor for com.tibco.wcc.components.ToolbarButton Component. The component name can be edited below.

wccToolbarButton

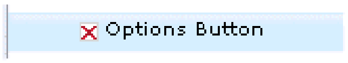
To display one of the available buttons, check the box for that button in the Properties Editor. For example, to display the **Options** button, do the following:



ToolbarButton - 'wccToolbarButton'
Component Model Name - 'wccToolbarButton'
Configuration Administrator Button
Logout Button
Open Next Work Item Button
✓ Options Button
Organization Browser Window Button

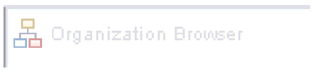
You can only display a single button in your application per Toolbar Button component. To include more than one of the toolbar buttons, place multiple Toolbar Button components in your application.

Also note that denying access to a button via the Properties Editor (by placing a red X in the box) has no effect when using the Toolbar Button component (which is different than how the other WCC components work):



Access to the functionality provided by the Toolbar Button component can only be allowed/denied using the user access sets and system actions. For information about user access sets and system actions, see the *TIBCO Workspace Configuration and Customization* guide.

If access to one of the toolbar buttons (with the exception of the **Logout** button, which has no access control) is denied (via user access sets), a grayed-out, disabled version of the button is displayed. For example:



Subscribe To:

To display the **Toolbar Button** component, subscribe to:

- **Login** component:
  - "Login Complete" event

Toolbar Button Properties

The **Toolbar Button** component has the following properties:

Property	Description
Configuration Administrator Button	Displays the **ConfigAdmin** button — see Organization Browser Window Button .
Logout Button	Displays the **Logout** button — see Logout Button .
Open Next Work Item Button	Displays the **Open Next Work Item** button — see Open Next Work Item Button .
Options Button	Displays the **Options** button — see Options Button .
Organization Browser Window Button	Displays the **Organization Browser** button — see Organization Browser Window Button .

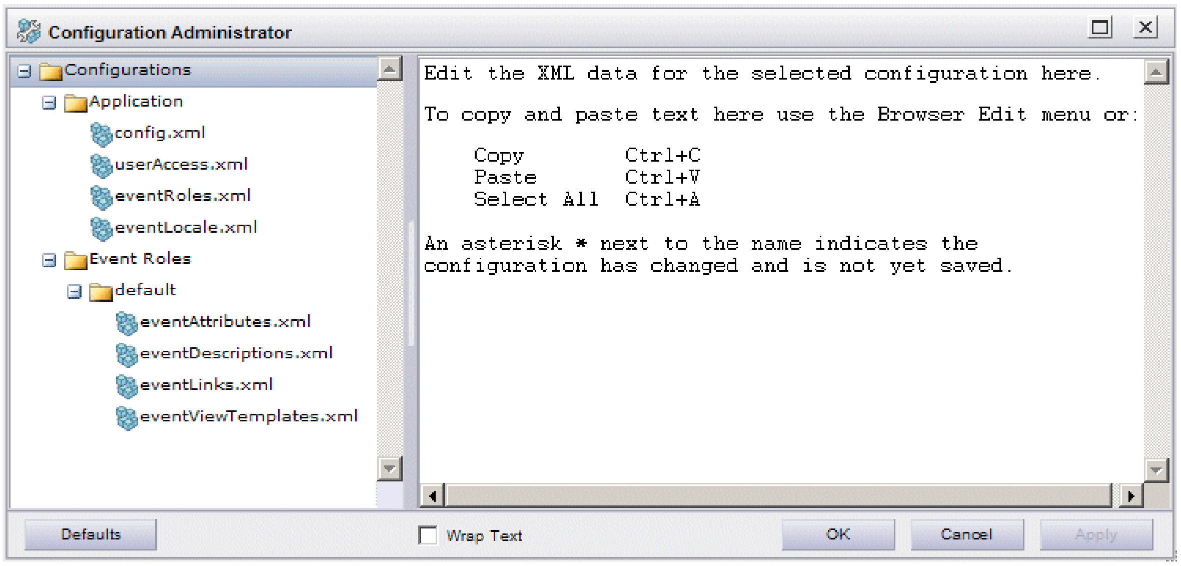
Toolbar Button Events

None.

Configuration Administrator Button

The **ConfigAdmin** button displays the Configuration Administrator dialog, which is used to configure a deployed application.

When the user clicks the **Configuration Administrator** button, the following dialog is displayed:



```
Edit the XML data for the selected configuration here.

To copy and paste text here use the Browser Edit menu or:

    Copy        Ctrl+C
    Paste       Ctrl+V
    Select All  Ctrl+A

An asterisk * next to the name indicates the
configuration has changed and is not yet saved.
```

To display the **Configuration Administrator** button in your WCC application:

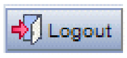
1. Place the **Toolbar Button** component in the desired location in your application.
2. In the Properties Editor, check the **Configuration Administrator Button** property.
3. In the Events Editor, subscribe to the event that you want to cause the button to be displayed (typically, Login Complete).
4. Commit your changes.

For more information, see Toolbar Button Component.

For information about using the Configuration Administrator, see the "Introduction" topic in the *TIBCO Workspace Configuration and Customization Guide.*

Logout Button

The **Logout** button logs the user out of the application.



To display the **Logout** button in your WCC application:

1. Place the **Toolbar Button** component in the desired location in your application.
2. In the Properties Editor, check the **Logout Button** property.
3. In the Events Editor, subscribe to the event that you want to cause the button to be displayed (typically, Login Complete).
4. Commit your changes.

For more information, see Toolbar Button Component.

Cleaning Up After a Logout

By default, clicking the **Logout** button logs the user out of the server, but it does *not* clean up the screen in any way. It is felt that every custom application will desired some custom logic upon a logout.

When the user clicks the **Logout** button, the **postLogout** function in the `Application.js` file is always called. To perform custom logic when the user logs out, add your custom code in the **postLogout** function.

The `Application.js` file is located in the following directory:

`*StudioHome*\wcc\*version*\JSXAPPS\*WCCProjectName*\application\js`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.
- *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".

Open Next Work Item Button

The **Open Next Work Item** button opens the next available work item in the work item list, where “available” means a work item that is not locked nor suspended. Typically, that is the first work item in the list.

If the user clicks the **Open Next Work Item** button, but there are no available work items, a "No available work items to open" message is displayed.

To display the **Open Next Work Item** button in your WCC application:

1. Place the **Toolbar Button** component in the desired location in your application.
2. In the Properties Editor, check the **Open Next Work Item Button** property.
3. In the Events Editor, subscribe to the event that you want to cause the button to be displayed (typically, Login Complete).
4. Commit your changes.

For more information, see Toolbar Button Component.

Options Button

The **Options** button displays the Options dialog, which is used to establish default settings for the currently logged-in user.

To display the **Options** button in your WCC application:

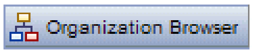
1. Place the **Toolbar Button** component in the desired location in your application.
2. In the Properties Editor, check the **Options Button** property.
3. In the Events Editor, subscribe to the event that you want to cause the button to be displayed (typically, Login Complete).
4. Commit your changes.

For more information, see Toolbar Button Component.

For information about using the Options dialog to set user options, see the "User Options" topic in the *TIBCO Workspace User’s Guide.*

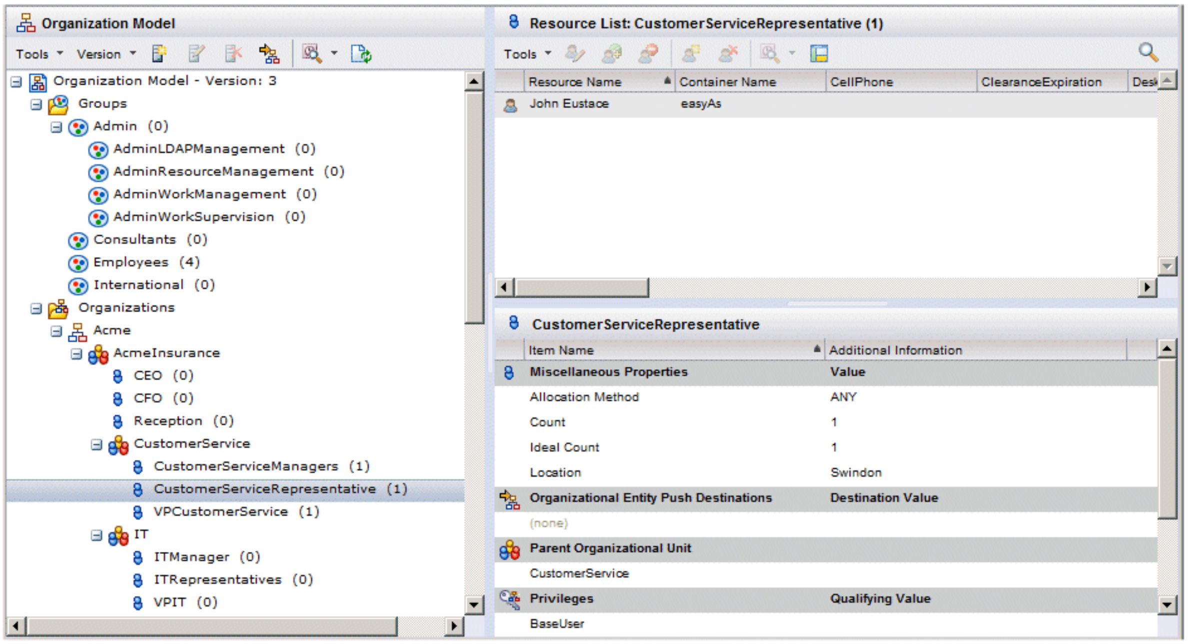
Organization Browser Window Button

The **Organization Browser** button displays the Organization Browser dialog, which is used to browse the organization model, create LDAP containers, map resources to groups/positions, etc.



When the user clicks the **Organization Browser** button, the following dialog is displayed:

The dialog displayed by the **Organization Browser** button, is the same dialog that is displayed by the Organization Resource Browser composite component.


Organization Model

Tools Version

- Organization Model - Version: 3
  - Groups
    - Admin (0)
      - AdminLDAPManagement (0)
      - AdminResourceManagement (0)
      - AdminWorkManagement (0)
      - AdminWorkSupervision (0)
    - Consultants (0)
    - Employees (4)
    - International (0)
  - Organizations
    - Acme
      - AcmeInsurance
        - CEO (0)
        - CFO (0)
        - Reception (0)
        - CustomerService
          - CustomerServiceManagers (1)
          - CustomerServiceRepresentative (1)
          - VPCustomerService (1)
        - IT
          - ITManager (0)
          - ITRepresentatives (0)
          - VPIT (0)

Resource List: CustomerServiceRepresentative (1)

Tools

Resource Name	Container Name	CellPhone	ClearanceExpiration	Desk
John Eustace	easyAs			

CustomerServiceRepresentative

Item Name	Additional Information
Miscellaneous Properties	**Value**
Allocation Method	ANY
Count	1
Ideal Count	1
Location	Swindon
Organizational Entity Push Destinations	**Destination Value**
(none)	
Parent Organizational Unit	
CustomerService	
Privileges	**Qualifying Value**
BaseUser	

To display the **Organization Browser** button in your WCC application:

1. Place the **Toolbar Button** component in the desired location in your application.
2. In the Properties Editor, check the **Organization Browser Window Button** property.
3. In the Events Editor, subscribe to the event that you want to cause the button to be displayed (typically, Login Complete).
4. Commit your changes.

For more information, see Toolbar Button Component.

For information about using the Organization Browser, see the *Organization Browser User's Guide.*

Organization Browser Components

The Organization Browser components are used to view the organization model, create LDAP containers, and to map resources in a container to groups and positions in the organization model so that they can log into your custom WCC application and receive work items in their work list.

The available Organization Browser components are:

- **Organization Browser** - This component displays a graphical representation of the organization model, and is used to create LDAP containers.
- **Organization Resource List** - This component displays a list of resources that have been added to the selected LDAP container, or that have been mapped to the selected organizational entity.

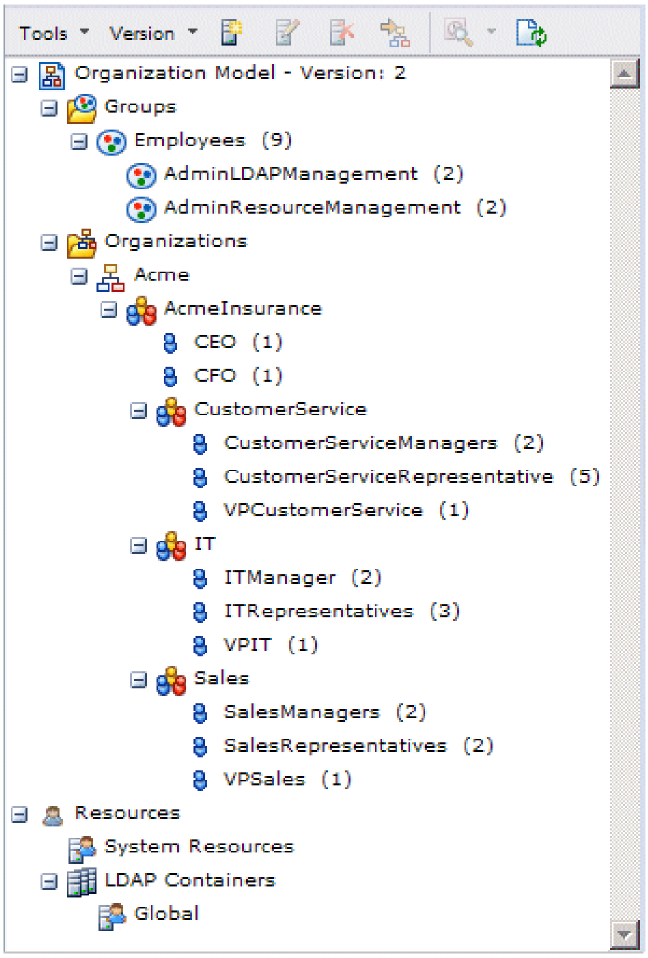
Also see the Organization Resource Browser component, which is a composite that includes both of the components listed above.

Organization Browser

This component can be used to browse the organization model. It displays a graphical representation of the organization model. It also allows the user to create LDAP containers, which contain resources that can be mapped to groups and positions in the organization model.

Typically, this component is used in conjunction with the Organization Resource List component — when the user selects an organizational entity in the **Organization Browser** component, the resources in that entity are displayed in the **Organization Resource List** component.

The Organization Browser component displays a dialog similar to the following:



```
Organization Model - Version: 2
  Groups
    Employees (9)
      AdminLDAPManagement (2)
      AdminResourceManagement (2)
  Organizations
    Acme
      AcmeInsurance
        CEO (1)
        CFO (1)
        CustomerService
          CustomerServiceManagers (2)
          CustomerServiceRepresentative (5)
          VPCustomerService (1)
        IT
          ITManager (2)
          ITRepresentatives (3)
          VPIT (1)
        Sales
          SalesManagers (2)
          SalesRepresentatives (2)
          VPSales (1)
Resources
  System Resources
  LDAP Containers
    Global
```

For information about the functions available from this component, see the *TIBCO Organization Browser User's Guide.*

Access to each of the functions available in the Organization Browser is controlled by properties on the **Organization Browser** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Organization Browser Component Properties.

The **Organization Browser** component also publishes events for actions executed from the Organization Browser. For information about these events, see Organization Browser Component Events.

Subscribe To:

To display the **Organization Browser** component, subscribe to:

- **Login** component:
  - "Login Complete" event

Organization Browser Component Properties

The **Organization Browser** component contains the following properties:

Property	Description
Organization Browser	Enables / disables the ability to access all of the functions in the Organization Browser, i.e., all of the functions described in this table.
Show Containers Tree	Enables / disables the ability to view LDAP containers. Controls whether or not the LDAP containers tree is displayed in the Organization Browser.
Show Groups Tree	Enables / disables the ability to view groups. Controls whether or not the groups tree is displayed in the Organization Browser.
Show Organization Tree	Enables / disables the ability to view organizations. Controls whether or not the organization tree is displayed in the Organization Browser.
Show Organization Event Viewer Links	Enables / disables the ability to create an event view containing organizational entity events. Controls access to the "**View Events**" button and menu selection in the Organization Browser.
Manage LDAP Containers	Enables / disables the ability to create, edit, and delete LDAP containers in the Organization Browser. Controls access to all of the LDAP container-related buttons and menu selections in the Organization Browser.
Manage LDAP Containers - New Container	Enables / disables the ability to create a new LDAP container in the Organization Browser. Controls access to the **New LDAP Container** button and menu selection in the Organization Browser.
Manage LDAP Containers - Edit Container	Enables / disables the ability to edit an existing LDAP container in the Organization Browser. Controls access to the **Edit LDAP Container** button and menu selection in the Organization Browser.
Manage LDAP Containers - Delete Container	Enables / disables the ability to delete an LDAP container in the Organization Browser. Controls access to the **Delete LDAP Container** button and menu selection in the Organization Browser.

Property	Description
Edit Organization	Enables / disables the ability to edit organizational entity push destinations, as well as export and import LDAP containers in the Organization Browser. Controls access to the **Edit Organizational Entity Push Destinations**, **Import LDAP Containers**, and **Export LDAP Containers** buttons and menu selections in the Organization Browser.
Edit Organization - Edit Org Push Destinations	Enables / disables the ability to edit organizational entity push destinations in the Organization Browser. Controls access to the **Edit Organizational Entity Push Destinations** button and menu selection in the Organization Browser.
Edit Organization - Import Containers / Resources	Enables / disables the ability to import LDAP containers in the Organization Browser. Controls access to the **Import LDAP Containers** button and menu selection in the Organization Browser.
Edit Organization - Export Containers / Resources	Enables / disables the ability to export LDAP containers in the Organization Browser. Controls access to the **Export LDAP Containers** button and menu selection in the Organization Browser.

Organization Browser Component Events

The **Organization Browser** component publishes the following events:

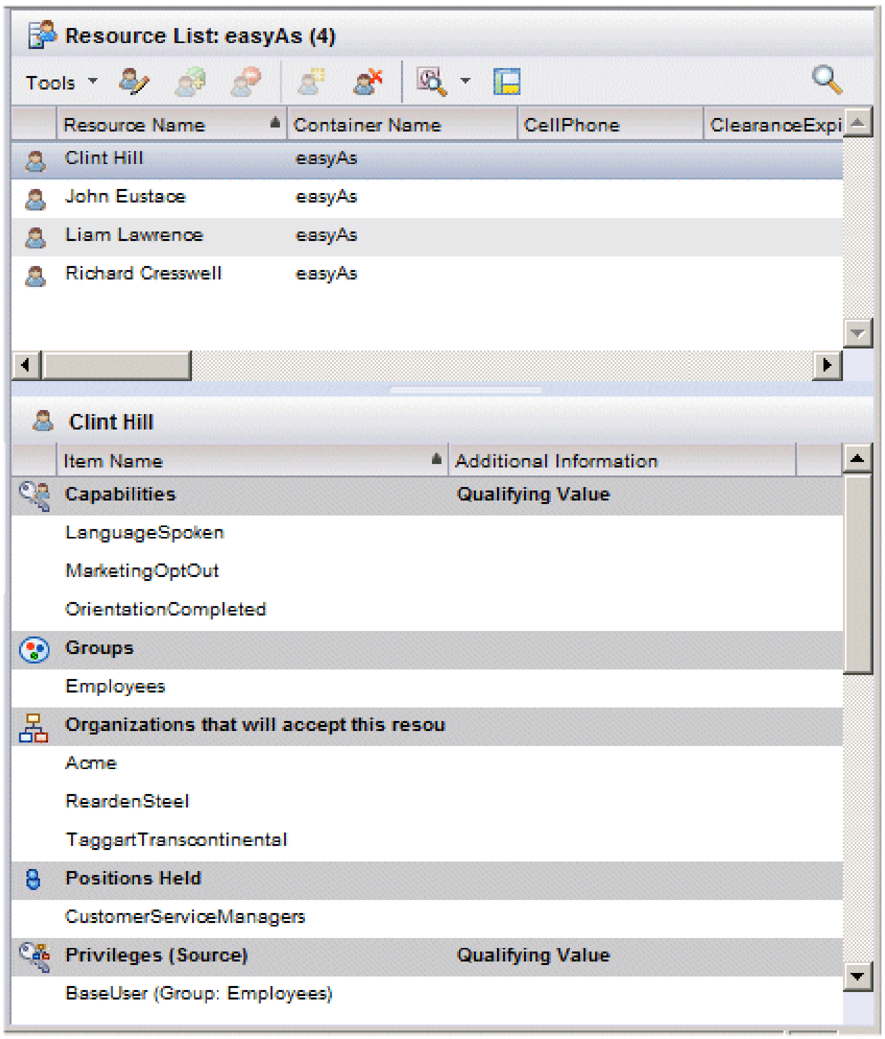
Event	Description
Organization Item Select	Indicates an organizational entity in the Organization Browser has been selected. Event fires when the user single-clicks on any organizational entity in the Organization Browser.
Show Events	Indicates the user is generating a list of events for the selected organizational entity. Event fires when the user selects the "**View Events**" button or menu selection in the Organization Browser.

Organization Resource List

This component displays a list of resources that have been added to the selected LDAP container, or that have been mapped to the selected organizational entity.

Typically, this component is used in conjunction with the Organization Browser component — when the user selects an organizational entity in the **Organization Browser** component, the resources in that entity are displayed in the **Organization Resource List** component.

The Organization Resource List displays a dialog similar to the following:


Resource List: easyAs (4)

Tools

Resource Name	Container Name	CellPhone	ClearanceExpi
Clint Hill	easyAs		
John Eustace	easyAs		
Liam Lawrence	easyAs		
Richard Cresswell	easyAs		

Clint Hill

Item Name	Additional Information
Capabilities	**Qualifying Value**
LanguageSpoken	
MarketingOptOut	
OrientationCompleted	
Groups	
Employees	
Organizations that will accept this resou	
Acme	
ReardenSteel	
TaggartTranscontinental	
Positions Held	
CustomerServiceManagers	
Privileges (Source)	**Qualifying Value**
BaseUser (Group: Employees)	

When the user selects a resource in the upper pane, the lower pane displays information about the resource, such as capabilities, privileges, resource attributes, and push destination, as well as the groups and positions to which the resource has been mapped.

For information about the functions available from this component, see the *Organization Browser User's Guide*.

Access to each of the functions available in the Organization Resource List is controlled by properties on the **Organization Resource List** component (in conjunction with user access sets, which are described in the *TIBCO Workspace Customization and Configuration* guide). For information about setting these properties, see Organization Resource List Component Properties.

The **Organization Resource List** component also publishes events for actions executed from the Organization Browser. For information about these events, see Organization Resource List Component Events.

Subscribe To:

To display the **Organization Resource List** component, subscribe to:

- **Organization Browser** component:
  - Organization Item Select

Organization Resource List Component Properties

The **Organization Resource List** component contains the following properties:

Property	Description
Organization Browser	Enables / disables the ability to access all of the functions in the Organization Resource List component, i.e., all of the functions described in this table. (If this property is disabled, all functions on the resource list are disabled, except for the Toggle Preview and Find functions, which do not have corresponding properties.) Note that this top-level property is called "Organization Browser" even though it is controlling access to the functions on the Organization Resource List component. That is because of the hierarchy of the user access sets, to which these properties have a one-to-one correspondence.
List Potential Resources	Enables / disables the ability to view potential resources. If enabled, *potential* resources are shown in the resource list (potential resources are those that are in the resource list, but have not been created nor mapped to a group or position; they are greyed out in the resource list).
Show Resource Event Viewer Links	Enables / disables the ability to create event views containing events related to the selected resource. Controls access to the "**View Events**" button and menu selection in the resource list.
Show Resource Attributes In Resource List	Enables / disables the ability to view resource attribute values in the resource list. Controls whether or not columns representing resource attributes are displayed in the resource list.
Show Organization Preview	Enables / disables the ability to view details of organizational entities. Controls whether or not organizational information is displayed in the resource list details pane when an organizational entity is selected in the Organization Browser.
Show Organization Preview - Preview Privileges	Enables / disables the ability to view organizational entity privilege information. Controls whether or not the privileges assigned to the selected organizational entity are displayed in the resource list details pane.
Show Organization Preview - Preview Capabilities	Enables / disables the ability to view organizational entity capability information. Controls whether or not the required capabilities for the selected organizational entity are displayed in the resource list details pane.

Property	Description
Show Organization Preview - Preview Push Destinations	Enables / disables the ability to view organizational entity push destinations. Controls whether or not the push destinations assigned to the selected organizational entity are displayed in the resource list details pane.
Show Resource Preview	Enables / disables the ability to resource detail information. Controls access to the details pane (lower-right pane) to view information (group and position membership, capabilities, etc.) for the selected resource.
Show Resource Preview - Preview Group Membership	Enables / disables the ability to view resource group membership. Controls whether or not group membership for the selected resource is displayed in the resource list details pane.
Show Resource Preview - Preview Position Held	Enables / disables the ability to view resource position-held information. Controls whether or not the positions held by the selected resource are displayed in the resource list details pane.
Show Resource Preview - Preview Resource Attributes	Enables / disables the ability to view a resource's resource attribute information. Controls whether or not resource attribute values for the selected resource are displayed in the resource list details pane.
Show Resource Preview - Preview Privileges	Enables / disables the ability to view a resource's privileges. Controls whether or not privileges for the selected resource are displayed in the resource list details pane. (Privileges are obtained by virtue of being mapped to a position or group that has that privilege.)
Show Resource Preview - Preview Capabilities	Enables / disables the ability to view a resource's capabilities. Controls whether or not capabilities assigned to the selected resource are displayed in the resource list details pane.
Show Resource Preview - Preview Push Destinations	Enables / disables the ability to view a resource's push destinations. Controls whether or not push destinations specified for the selected resource are displayed in the resource list details pane.
Edit Resources	Enables / disables the ability to edit resource information. Controls access to all of the edit-type functions on the Organization Resource List component.

Property	Description
Edit Resources - Edit Group Membership	Enables / disables the ability to edit group membership for the selected resource(s). If this access is denied, and not the "EditPositionsHeld" access (see below), the Group and Position Assignment Editor dialog is still displayed when the **Edit Groups and Positions Held** function is selected. However, the buttons that allow you to grant or remove group membership for the selected resource(s) are not displayed. If this access *and* the "EditPostionsHeld" access are both denied, the **Edit Groups and Positions Held** function is not available, i.e., you cannot access the Group and Position Assignment Editor dialog.
Edit Resources - Edit Positions Held	Enables / disables the ability to edit the positions held by the selected resource(s). If this access is denied, and not the "EditGroupMembership" access (see above), the Group and Position Assignment Editor dialog is still displayed when the **Edit Groups and Positions Held** function is selected. However, the buttons that allow you to grant or remove positions held for the selected resource(s) are not displayed. If this access *and* the "EditGroupMembership" access are both denied, the **Edit Groups and Post ions Held** function is not available, i.e., you cannot access the Group and Position Assignment Editor dialog.
Edit Resources - Edit Resources Attributes	Enables / disables the ability to edit the resource attributes for the selected resource(s). Controls access to the **Edit Resource Attributes** button and menu selection in the Organization Resource List component.
Edit Resources - Edit Capabilities	Enables / disables the ability to edit the capabilities for the selected resource(s). Controls access to the **Edit Capabilities** button and menu selection in the Organization Resource List component.
Edit Resources - Edit Push Destinations	Enables / disables the ability to edit the push destinations for the selected resource. Controls access to the **Edit Resource Push Destination** button and menu selection in the Organization Resource List component.
Edit Resources - Create Resources	Enables / disables the ability to create a resource. Controls access to the **Create Resources** button and menu selection in the Organization Resource List component.
Edit Resources - Delete Resources	Enables / disables the ability to delete a resource. Controls access to the **Delete Resources** button and menu selection in the Organization Resource List component.

Organization Resource List Component Events

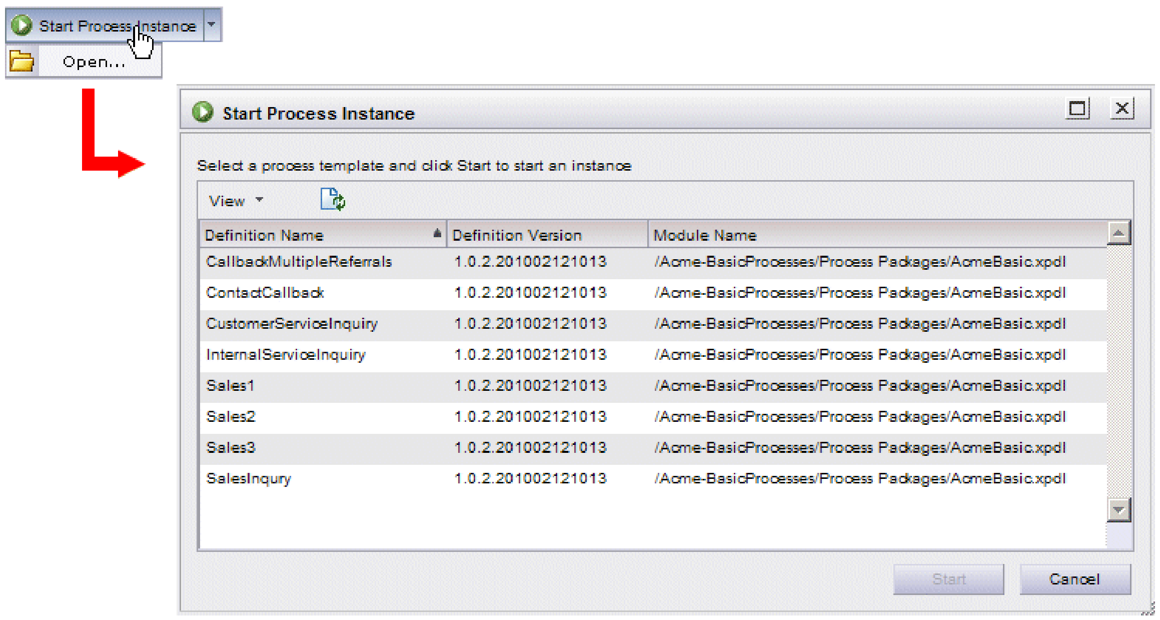
The **Organization Resource List** component publishes the following events:

Event	Description
Resource Item Select	Indicates a resource in the resource list has been selected. Event fires when the user single-clicks on any resource in the Organization Resource List component.
Show Events	Indicates the user is generating a list of events for the selected resource. Event fires when the user selects the "**View Events**" button or menu selection in the resource list.

Start Instance Component

The **Start Instance** component displays the **Start Process Instance** button, which allows the user to start an instance of a process template from a list of available process templates.

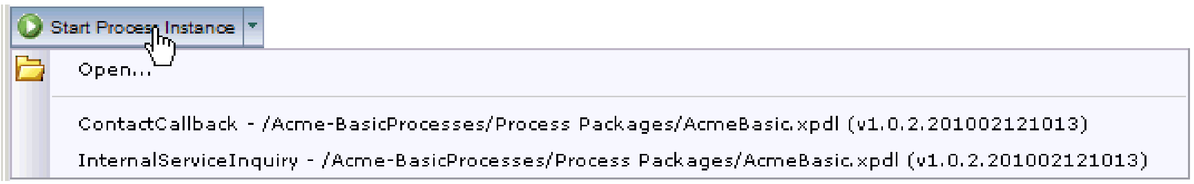
If the user clicks on the **Start Process Instance** button, then selects the **Open** drop-down selection, a list of available process templates is displayed. For example:



Select a process template and click Start to start an instance

Definition Name	Definition Version	Module Name
CallbackMultipleReferrals	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
ContactCallback	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
CustomerServiceInquiry	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
InternalServiceInquiry	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
Sales1	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
Sales2	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
Sales3	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl
SalesInqury	1.0.2.201002121013	/Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl

Once the user has made a selection from the list of process templates, subsequent clicks of the **Start Process Instance** button also displays the previously started process templates, making it easier to choose process templates that are frequently started. For example:



ContactCallback - /Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl (v1.0.2.201002121013)
InternalServiceInquiry - /Acme-BasicProcesses/Process Packages/AcmeBasic.xpdl (v1.0.2.201002121013)

Selecting a process template from either the list of all available templates, or from the button drop-down list, causes an instance of that template to start.

By default, the number of available process templates that can be listed is limited to 500. To increase this number, change the value of the property com.tibco.bx.management.queryMaxResultSize, as explained in the *TIBCO ActiveMatrix BPM Administration* guide.

Subscribe To:

To display the **Start Instance** component, subscribe to:

- **Login** component:
  - "Login Complete" event

Start Instance Component Properties

None.

Start Instance Component Events

The **Start Instance** component publishes the following event:

Event	Description
InstanceStarted	Indicates an instance of a process template has been started. Event fires when the user selects a process template to start.

WCC Public Methods

Public methods are provided that allow you to add custom business logic, as well as programmatically perform functions that are normally performed by WCC components, for example, opening a work item, cancelling a process instance, etc.

For information about the classes that are created (that provide access to the public methods), when you develop a custom WCC application, see Application Template.

Note that all files under the following directories are considered to be TIBCO's private implementation of WCC components and should not be modified:

```
JSXAPPS\base
```

```
JSXAPPS\components
```

There are some exceptions to this rule for the `\JSXAPPS\base` directory: you are allowed to modify the files in the `\JSXAPPS\base\locale\eventRoles` directory (see the "Configuring Events" topic in the *TIBCO Workspace Configuration and Customization* guide), and in the `\JSXAPPS\base\jss\workspaceCSS.xml` file (see the "Customizations" topic in the *TIBCO Workspace Configuration and Customization* guide), as well as in the `\JSXAPPS\base\locale\locale.xml` file (to localize strings displayed in the application).

Only files under the following directory should be modified:

```
JSXAPPS\wccApp
```

where *wccApp* is the name of the custom WCC application (which is “workspace” if you are using the Workspace application).

The following comprise the public methods that can be used in custom WCC applications:

- **Override methods** - These methods, which provide entry points into your WCC application for custom code, are located in the main Application class (`Application.js`) for your custom WCC application.

  For details about the available override methods, see Override Methods.

- **Application class methods** - These methods can be called from your custom application to perform various functions like getting the application block, adding an entry to the application log, etc. They also provide the means to perform functions normally performed by WCC components, such as logging in, opening a work item, etc.

  For details about the available Application class methods, see Application Class Methods.

- **Mixin interface methods** - The mixin interface (`AppMain.js`) is implemented by the Application class for your custom application. It contains samples of how custom methods can be added. Plus, it provides a container for other custom code you want to add.

  For details about the method examples in the mixin interface, see Mixin Interface Methods.

- **Component public methods** - A number of the WCC components contain public methods that allow you to perform functions specific to that component. For instance, the WorkItemsPreview component contains public methods that allows you to hide or show text in the caption bar of the WorkItemsPreview component.

  These methods must be called from their respective component instance (e.g., com.tibco.wcc.components.WorkItemsPreview).

  For details about the component public methods, see Component Public Methods.

- **Prototype Public Methods** - The com.tibco.wcc.components.Prototype class contains a single public method, getVersion.

  For information, see Prototype Public Methods.

- **Server Requests** - An API is available that allows you to make direct calls to the server to perform tasks such as listing work items, allocating work items, suspending process instances, etc.

  For information, see Server Requests.
- **Callout Interface Methods** - The methods in the callout interface can be used to specify filter expressions, sort parameters, available filter attributes, available sort attributes, and display columns for various lists in WCC applications (including the Workspace application).

  For information, see Callout Interface Methods.

Override Methods

The override methods provide entry points to add custom code into your WCC application.

These methods are located in the main Application-level class, `Application.js`, which is located in the following directory (note that they should not be called directly from the extending application):

`StudioHome\wcc\version\JSXAPPS\WCCProjectName\application\js\`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.
- *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".

The following are the available override methods:

- **init** - This method is called when the application instance is constructed. The extending Application class can add functionality as required by overriding this method.
- **postLoadInit** - This method is called after the initial prototype file is loaded and painted. The extending Application class can add functionality as required by overriding this method.
- **postLogin** - This method is called by the Login component after the login has been successfully completed, but before publishing the loginComplete event. The extending Application class can add functionality as required by overriding this method.
- **preLogout** - This method is called by the Logout component prior to making a server call to logout. The extending Application class can add functionality as required by overriding this method.
- **postLogout** - This method is called by the Logout component after the logout has been successfully completed. The extending Application class can add functionality as required by overriding this method.
- **onBeforeUnload** - This method is called before the browser page is closed. It can be used to handle any kind of actions that should be performed before the browser's **onbeforeunload** event is fired.
- **onUnload** - This method is called when the browser page is closed. It can be used to handle any kind of actions that should be performed before the browser's **onunload** event is fired.
- **loadCustomMenu** - This method loads a custom menu or toolbar button if a definition is found in `config.xml`.
- **onSessionInvalid** - This method handles any actions that should be performed when the current login session is no longer valid.

init

This method is called when the application instance is constructed.

Your extending Application class can override this method, and include custom functionality as required.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

Syntax

```
init(appRoot);
```

Parameters

- *appRoot* - (jsx3.gui.Block) The root of the application. This is typically the JSXBODY jsx3.gui.Block instance; it is determined by the **appRoot** value in the `wccConfig.xml` file. If **appRoot** is not specified in `wccConfig.xml`, it is set to the body block returned by **server.getBodyBlock()**.

  For information about the `wccConfig.xml` file, see the *TIBCO Workspace Configuration and Customization* guide.

Also see: getAppBlock

Returns

none

postLoadInit

This method is called after the initial prototype file (which is specified in the **path** attribute of the **mainPrototype** element in the `wccConfig.xml` file) is loaded and painted. This is only called if the **mainPrototype** element exists and its **path** attribute is specified.

Your extending Application class can override this method, and include custom functionality as required.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

Syntax

```
postLoadInit(path);
```

Parameters

- *path* - (String) The path to the main prototype; this is the path specified in the **path** attribute of the **mainPrototype** element in `wccConfig.xml` file.

  For information about the `wccConfig.xml` file, see the *TIBCO Workspace Configuration and Customization* guide.

Returns

none

postLogin

This method is called by the Login component after the login has been successfully completed, but before publishing the loginComplete event.

It performs the following base-level post-login actions:

- Sets the window caption.

- Loads options data.
- Subscribes to application-level events.

Your extending Application class can override this method, and include custom functionality as required.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

Syntax

```
postLogin();
```

Parameters

none

Returns

none

preLogout

This method is called by the Logout component prior to making a server call to logout. It can be used to handle any kind of actions that should be performed before logout.

Your extending Application class can override this method, and include custom functionality as required.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

Syntax

```
preLogout();
```

Parameters

none

Returns

none

postLogout

This method is called by the Logout component after the logout has been successfully completed. It can be used to handle any kind of actions that should be performed after logout.

Your extending Application class can override this method, and include custom functionality as required.

Note that when a session timeout warning message is displayed (see the **SessionMonitor** parameter in the *TIBCO Workspace Configuration and Customization* guide), a blue mask is displayed behind the warning message, covering the application window. This ensures that application data is not visible if the workstation is left unattended and a session timeout is about to occur. When the session timeout occurs, it is up to the custom application to determine what to display upon logout. Logic should be added to the **postLogout** method to reset the application similarly to the way it is done in the customPostLogoutExample function.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

Syntax

```
postLogout();
```

Parameters

none

Returns

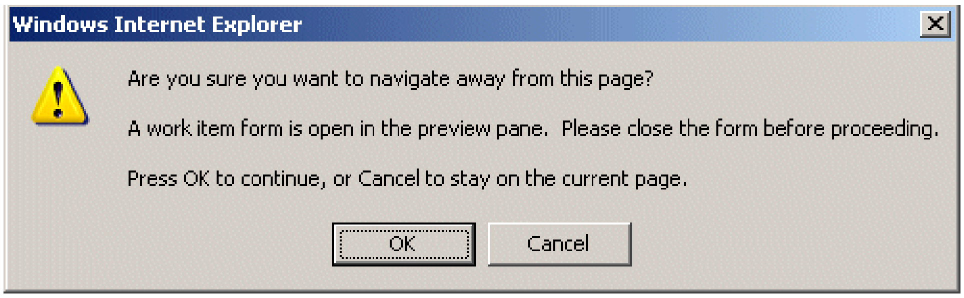
none

onBeforeUnload

This method is called before the browser page is closed. It can be used to handle any kind of actions that should be performed before the browser's **onbeforeunload** event is fired.

Your extending Application class can override this method, and include custom functionality as required.

If a value other than 'undefined' is returned from this method, the browser displays a confirmation dialog containing a message like the following:


Windows Internet Explorer

Are you sure you want to navigate away from this page?

A work item form is open in the preview pane. Please close the form before proceeding.

Press OK to continue, or Cancel to stay on the current page.

The extending method may call the **this.jsxsuper()** method to include the base functionality. The base method calls **isWorkItemPreviewFormOpen(false)** to check for any work items open in a preview pane. If so, it returns the value of the dynamic property, **txtWorkItemChanged**.

Syntax

```
onBeforeUnload();
```

Parameters

none

Returns

- undefined or string
  - if undefined, the browser window is closed, unless there is another handler that returns a value other than undefined.
  - if a string, it is displayed in the confirmation dialog.

onUnload

This method is called when the browser page is closed. It can be used to handle any kind of actions that should be performed before the browser's **onunload** event is fired.

Your extending Application class can override this method, and include custom functionality as required.

The extending method may call the **this.jsxsuper()** method to include the base functionality. The base method calls the following to logout before the browser page is closed:

```
var maskMsg = this.getServer().
getDynamicProperty('txtOnUnloadLogout');
logoutOnUnload(this, true, maskMsg)
```

Syntax

```
onUnload();
```

Parameters

none

Returns

none

loadCustomMenu

This method loads a custom menu or toolbar button if a definition is found in the **customMenus** parameter in the `config.xml` file.

This method is called by various WCC components that allow custom menus and toolbar buttons (those that are represented by the valid values for the *parentId* parameter — see below).

Your extending Application class can override this method, and include custom functionality as required.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

This method can also be overridden by the custom application to check for authority to access the custom menu and toolbar button. See the **loadCustomMenu** method in the following file for an example of using the **isAuth** method to check authority:

```
JSXAPPS\wccApp\application\js\application\Application.js
```

where *wccApp* is the name of your custom WCC application.

For more information about adding custom menus and toolbar buttons to your custom application, see the *TIBCO Workspace Configuration and Customization* guide.

Also see: getCustomMenuNode

Syntax

```
loadCustomMenu(type,
               parentId,
               container);
```

Parameters

- *type* - (String) Identifies whether it's a custom menu or toolbar button. The valid values are:

  - "menu"
  - "toolbar"
- *parentId* - (String) Identifies where the custom menu or toolbar is to be added. This value must match the value in the **customMenus** parameter in the `config.xml` file. The valid values are:
  - "AuditEventList"
  - "BusinessServiceList"
  - "EventViewList"
  - "ProcessInstanceList"
  - "ProcessTemplateList"
  - "ProcessViewList"
  - "WorkItemList"
  - "WorkViewList"

  Also note that a custom WCC application can set other values by calling **getCustomMenuNode** on the WCC application instance and providing its own handling for adding the menu or toolbar specified by the node. For more information, see getCustomMenuNode.
- *container* - (Object) The parent jsx3.gui object to load the custom menu or toolbar into.

Returns

none

onSessionInvalid

This method handles any actions that should be performed when the current login session is no longer valid. This is usually due to a session timeout or opening another browser instance of this application on the same session.

The base class implementation for this method displays the txtInvalidLogin message to the user and reloads the application using:

```
window.location.reload(false);
```

Your extending Application class can override this method, and include custom functionality as required.

The extending method must call the following method to include the functionality provided in the base Application class:

- this.jsxsuper();

Syntax

```
onSessionInvalid ();
```

Parameters

none

Returns

none

Application Class Methods

The Application class methods can be called from your custom application to perform various functions. Your custom application has access to these methods by virtue of it extending `com.tibco.wcc.components.Application`, the "base" Application class.

Only the methods that are listed here are public; all others are considered private and are subject to change.

General Application Class Methods

- **getApp** - Returns the instance of the com.tibco.wcc.components.Application object for the specified namespace. Note that this is a static method; also see the getApp method, which returns the current application instance.
- **getAppVersion** - Returns the version of the current build.
- **getProductBrand** - Returns the **"**brand**"** of the product: "ActiveMatrix" or "Silver".
- **logoutOnUnload** - Logs out of the given server.
- **getNamespace** - Returns the application namespace.
- **getModelName** - Returns the application model name.
- **getPageBus** - Returns the PageBus instance.
- **isWorkItemPreviewFormOpen** - Returns a boolean value indicating whether or not the work item form is open in the preview pane.
- **getApp** - Returns the current application instance. Also see the **getApp** method on getApp, which returns the application instance for the given namespace.
- **getAppBlock** - Returns the top-level application block.
- **getServer** - Returns the jsx3.app.Server for the application.
- **isAuth** - Returns a boolean value indicating whether or not the user is authorized for the given profileCode.
- **clearCacheData** - Clears documents from the application cache.
- **publishPauseWorkItemsAutoRefresh** - Used to either pause or un-pause the auto-refresh of work items in a work item list.
- **loadPrototype** - Loads a component prototype as a child of the given component object and calls paintChild to render the object.
- **adjustSpanForDialog** - Adjusts the position attributes of a block caption bar HTML span element to display correctly in a dialog jsx3.gui.WindowBar caption.
- **loadDialogPrototype** - Loads the dialogPrototype as a child of the parent container and centers the dialog on the window.
- **getCustomMenuNode** - Returns the XML node (jsx3.xml.Entity), which can be used by custom WCC applications when including custom menus and toolbar buttons.
- **registerLongLivedProcess** - Registers a long-lived process.
- **cancelLongLivedProcess** - Cancels a long-lived process that was previously registered with the registerLongLivedProcess method.
- **isExternalLogin** - Indicates whether or not there has been an external login.
- **getLocaleKey** - Returns the current Options setting for localeKey.
- **getBaseUrl** - Returns the base URL to the Action Processor.
- **getUsername** - Returns the user name of the logged-in user.

- **getUserGuid** - Returns the GUID of the logged-in user.
- **getChannelId** - Returns the Channel ID used to locate work item forms.
- **setChannelId** - Sets the value of the Channel ID used to locate work item forms.
- **getBrowserFeatures** - Returns an object with properties that specifies features of browser windows in which work items are displayed.
- **getSystemActionsAuthorized** - Returns an object with properties that represent system actions and their values for the logged-in user.
- **getUserAccessAuthorized** - Returns an object with properties that represent user access controls and their values for the logged-in user.
- **getPrivilegeNames** - Returns the privileges held by the logged-in user.
- **loadUrlArgs** - Returns an object with properties for each argument in the URL of the currently loaded page.

Data Mask Methods

- **isMaskActive** - Returns a boolean value indicating whether or not the “Loading data...” data mask is currently displayed in the application.
- **showMask** - Publishes a **showDataMask** PageBus event, and sets the maskActive property to true.
- **showMaskFunction** - Publishes a **showDataMask** PageBus event, sets the maskActive property to true, and calls the specified function asynchronously.
- **hideMask** - Publishes a **hideDataMask** PageBus event, and sets the maskActive property to false.

Options Methods

- **getOptionValue** - Returns the value of the specified Option property.
- **getOptionNames** - Returns the available Option property names.

Application Log Methods

- **appLogZeroTimer** - Resets the log base-line timer.
- **log** - Adds a new Application Log entry at the INFO level.
- **logDebug** - Adds a new Application Log entry at the DEBUG level.
- **logTrace** - Adds a new Application Log entry at the TRACE level.
- **logFatal** - Adds an Application Log entry at the FATAL level.
- **logError** - Adds an Application Log entry at the ERROR level.
- **logWarn** - Adds an Application Log entry at the WARN level.
- **logInfo** - Adds an Application Log entry at the INFO level.
- **getAppLogLevel** - Returns an integer representing the current log level.
- **getAppLogLevelName** - Returns the current log level name.
- **setAppLogLevel** - Sets the current log level using an integer.
- **setAppLogLevelName** - Sets the current log level using a log-level name.
- **isEchoToJsxLog** - Returns True if Application Log entries are echoed to the Application Monitor.
- **setEchoToJsxLog** - Specifies whether Application Log entries are echoed to the Application Monitor.
- **getJsxLogger** - Returns the logger used to write messages to the Application Monitor if **echoToJsxLog** is True.

- logException - Adds a new Application Log entry and exception object at the ERROR level and returns the resulting message string.
- logAlways - Specifies to always write passed message to both the Application Log and **jsx3.log** at the INFO level, regardless of what is configured for **appLogLevel** and **echoToJsxLog**.
- logStackTrace - Adds a new Application Log entry and a stackTrace to both the Application Log and **jsx3.log**.
- logHierarchy - Adds a new Application Log entry for the hierarchy of the given object to both the Application Log and **jsx3.log**.
- logTree - Adds a new Application Log entry for a tree view of children of the given object to both the Application Log and **jsx3.log**.
- logStart - Adds the given message as a timed Application Log entry.
- logStop - Sets the elapsed time for a timed Application Log entry with the matching specified message.
- getAppLogText - Returns the value of the Application Log text.
- clearAppLog - Clears all except the original initialization entries, then resets the Application Log baseline time to an effective zero elapsed time.

User Data Methods

- getUserDataAttributes - Gets persisted user data, then sets the properties of the specified object to the user data values.
- setUserDataAttributes - Sets persisted user data using the values in the properties of the specified object.

User Message Methods

- showUserMessage - Displays the given message in an alert.
- showErrorMessage - Displays the given error message in a dialog.
- confirmUserMessage - Displays the given message in a confirm window.

Display Caption Methods

- setCaptionBar - Displays an image and text on the specified bar.
- setCaptionBarText - Sets the display caption on the specified bar with the specified text.

Exception Handling Methods

- setToolsErrorHandlingEnabled - Sets the value of the **toolsErrorHandlingEnabled** property, which determines how to handle exceptions returned by the Action Processor.
- isToolsErrorHandlingEnabled - Returns the current value of the **toolsErrorHandlingEnabled** property.

Tools Interface Methods

The methods listed below are considered the *Tools Interface* methods — they differ from the other Application-class methods in that they make calls to the Action Processor. Because of that, exceptions are handled differently. For more information, see Tools Interface Methods.

- Access Methods
  - login - Issues a login request for the specified user.

- logout - Issues a logout request for the current user.
- initializeExternalLogin - Initializes the application after login is completed by an external application.

- Work Item Methods
  - allocateWorkItemToOfferSet - Allows a user to allocate one or more work items to a user from the original offer set.
  - allocateWorkItemToWorld - Allows a user to allocate one or more work items to any available user.
  - allocateWorkItemToSelf - Allows a user to allocate one or more work items to themselves.
  - cancelWorkItem - Closes the specified work item(s), discarding any changes that had been made.
  - openNextWorkItem - Opens the next available work item.
  - openNextWorkItemEx - Opens the next available work item, based on the specified filter and sort parameters.
  - openWorkItem - Opens the specified work item(s).
  - reofferWorkItem - Re-offers the specified work item(s) to the users to whom they were originally offered.
  - skipWorkItem - Marks the specified work item(s) as complete, removes them from the work item list, and causes the process to advance.
  - pendWorkItem - Causes the specified work item(s) to be hidden in the work item list until a specified date/time, or period of time has expired.
  - prioritizeWorkItem - Alters the priority of the specified work item(s).
- Process Instance Methods
  - startProcessInstance - Starts an instance of the specified process template.
  - cancelProcessInstance - Cancels the specified process instance(s), which results in associated work items also being cancelled.
  - resumeProcessInstance - Resumes the specified suspended process instance(s).
  - suspendProcessInstance - Suspends the specified process instance(s), resulting in associated work items also becoming suspended.
  - resumeHaltedProcessInstance - Resumes the specified halted process instance(s) so that the instance itself goes into a failed state because of the failed task.
  - retryProcessInstance - Retries the specified halted process instance(s) so that the process instance can progress normally.
  - ignoreProcessInstance - Causes the failed task to be skipped in the halted process instance. The process instance continues processing from the point in the process after the failed task.
- Business Service Methods
  - startBusinessService - Starts a business service.
- View Methods
  - showEvents - Adds a temporary view to the event view list.
- User Options Methods
  - workspaceOptions - Allows the user to view/modify his user options.

- Organization Browser Methods
  - **workspaceOrganizationBrowserWindow** - Displays the Organization Browser, which can be used to create LDAP containers, as well as map resources to groups and positions in the Organization Model.
- Configuration Methods
  - **configAdmin** - Displays the Configuration Administrator, which is used to configure a deployed application.
- Locale Methods
  - **getAppLocales** - Returns an array of jsx3.util.Locale objects, one for each locale supported by the application.
  - **setAppLocale** - Sets the locale of the application server.

General Application-Class Methods

The general Application-class methods provide a variety of functions used to customize your WCC application.

getApp

Returns the instance of the com.tibco.wcc.components`.Application` object for the specified namespace.

Note that this is a static method; also see the **getApp** method on getApp, which returns the current application instance.

If a component is passed in that is not attached to the DOM, (i.e., `namespaceRef.getServer() = null`), this method returns null.

Syntax

```
com.tibco.wcc.components.Application.getApp(namespaceRef);
```

Parameters

- *namespaceRef* - (String or Object) This is a unique identifier for the `jsx3.app.Server` instance. This can be one of the following:
  - String
  - Object - This must implement `getServer()`, which returns `jsx3.app.Server`.

Returns

- `com.tibco.wcc.components.Application`
- null if no matching application instance is found.

getAppVersion

Returns the version of the current Workspace build. This is the version and build number displayed on the “About Workspace” page.

An example is:

```
1.1.0 Build: 2011-02-03 07:00:00
```

Note: the “Build:” part of the string returned is localized with the **txtBuild** value in the `locale.xml` file.

Syntax

```
getAppVersion();
```

Parameters

None

Returns

- String

getProductBrand

Returns the "brand" of the product: "ActiveMatrix" or "Silver".

Syntax

```
getProductBrand();
```

Parameters

None

Returns

- String

logoutOnUnload

Logs the user out of the session for the special case where the browser page is being unloaded. An optional mask message can be specified to be displayed while the logout is being processed.

If an external login has been used ("**?externalLogin=true**" URL argument or **app.initializeExternalLogin** is called directly), this call has no effect and the previous session is not cleared.

For normal logouts, use the logout method.

Syntax

```
logoutOnUnload(showmask,
               maskMsg);
```

Parameters

- *showMask* - (boolean) (optional) If true (the default), a mask is shown on the main application block returned by app.getAppBlock(). If false, a mask is not shown.
- *maskMsg* - (String) (optional) A string value that is displayed in the mask if *showMask* is True. The default is an empty string.

Returns

None

getNamespace

Returns the value of the application **namespace** property.

The namespace value is specified in either the `config.xml` record `jsxid="namespace"`, or in the **jsxappns** attribute in the launch script.

Syntax

```
getNamespace();
```

Parameters

None

Returns

- String

getModelName

Returns the value of the application **modelName** property.

The modelName value always begins with `com.tibco.wcc`. This is followed by one of two values:

- The **appModelName** element, **name** attribute in `wccConfig.xml`.
- If the **appModelName** element, **name** attribute is not set in `wccConfig.xml`, the modelName value is based on the application **namespace** property value.

For information about the `wccConfig.xml` file, see the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
getModelName();
```

Parameters

none

Returns

- String

getPageBus

Returns the PageBus instance. This is the WCC wrapper to the window-scoped PageBus `window.PageBus`.

For more information, see the *TIBCO PageBus Developer's Guide*.

Syntax

```
getPageBus();
```

Parameters

none

Returns

- `com.tibco.wcc.components.PageBus`

isWorkItemPreviewFormOpen

Returns true if a work item form is open in a **WorkItemsPreview** component preview pane.

Syntax

```
isWorkItemPreviewFormOpen(showMsgIfOpen);
```

Parameters

- *showMsgIfOpen* - (boolean) If true, and a work item is open in the preview pane, a warning message is displayed indicating the condition. The message used is the value of the dynamic property, **txtWorkItemChanged**.

Returns

- boolean

getApp

Returns the current application instance.

Also see the **getApp** method on getApp, which returns the application instance for the given workspace.

Syntax

```
getApp();
```

Parameters

none

Returns

- `com.tibco.wcc.components.Application`

getAppBlock

Returns the top-level application block. This is the JSXBODY returned by `server.getBodyBlock()`, unless another component is specified in the **appRoot** element of `wccConfig.xml`.

For information about the `wccConfig.xml` file, see the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
getAppBlock();
```

Parameters

none

Returns

- `jsx3.gui.Block`

getServer

Returns the `jsx3.app.Server` for the application.

Syntax

```
getServer();
```

Parameters

none

Returns

- `jsx3.app.Server`

isAuth

Checks the authorization for the given **profileCode**. Returns true if authorized, false if not authorized.

The profileCode corresponds to the **access** element **name** attribute values in the following:

- **UserAccessSets** in `userAccess.xml` - Used when user access sets are being used to specify access privileges.
- **AccessDefaults** in `config.xml` - Used to specify default access privileges when user access sets are not being used.

The nesting of the elements is represented in the profileCode as the **name** values separated by a '.' character. Consider the following excerpt from the `userAccess.xml` file:

```
<access name="WorkView">
   <access name="SupervisedWorkItem">
      <access name="Cancel"/>
      <access name="Skip"/>
      <access name="Reoffer"/>
      <access name="AllocateToAnother">
               .
               .
               .
```

To determine the authorization for re-offering work items from a supervised work view, pass the following for *profileCode*:

- WorkView.SupervisedWorkItem.Reoffer

Syntax

```
isAuth(profileCode);
```

Parameters

- *profileCode* - (String) Identifies the function for which you are determining authorization. This consists of the name attributes as described above, separated by dot (.) characters.

Returns

- boolean

clearCacheData

Clears documents from the application cache that have a cache ID that contains the given *idString*.

Syntax

```
clearCacheData(idString);
```

Parameters

- *idString* - (String) Identifies the application cache. This is typically the ID of the object whose cache data is to be removed.

Returns

none

publishPauseWorkItemsAutoRefresh

Publishes a PageBus event that can be used to either pause or un-pause the auto-refresh of work items in a work item list.

The method publishes the following event:

```
com.tibco.wcc.appModelName.wccPrototype.wccComponent.
pauseWorkItemsAutoRefresh
```

The WorkItems component subscribes to the 'pauseWorkItemsAutoRefresh' event.

Syntax

```
publishPauseWorkItemsAutoRefresh(pauseState);
```

Parameters

- *pauseState* - (boolean) True to pause auto-refresh, False to un-pause auto-refresh.

Returns

none

loadPrototype

Loads a component prototype as a child of the given component object and calls **paintChild** to render the object.

Syntax

```
loadPrototype(parentComp,
              prototypePath);
```

Parameters

- *parentComp* - (jsx3.app.Model) The parent component to load the prototype into.
- *prototypePath* - (String) The relative path to the prototype XML.

Returns

- jsx3.app.Model - The instance of the deserialized component.

adjustSpanForDialog

Adjusts the position attributes of a block caption bar HTML span element to display correctly in a dialog jsx3.gui.WindowBar caption.

Syntax

```
adjustSpanForDialog(blockCaptionText);
```

Parameters

- *blockCaptionText* - (String) The caption text from a block caption bar that was initially set using the setCaptionBar method.

Returns

- String - The caption text with span adjusted for dialog bar.

loadDialogPrototype

Loads the dialog identified by the *dialogPrototype* parameter as a child of the parent container and centers the dialog on the window.

Syntax

```
loadDialogPrototype(parent,
                    dialogPrototype);
```

Parameters

- *parent* - (jsx3.gui.Block) The parent container.
- *dialogPrototype* - (String) The path to the `dialog.xml` file.

Returns

- jsx3.gui.Dialog - The loaded dialog instance.

getCustomMenuNode

Returns the XML node (jsx3.xml.Entity), which can be used by custom WCC applications when including custom menus and toolbar buttons.

Custom menus and toolbar buttons can be added to the standard components (WorkItemList, ProcessInstanceList, etc.) using the **customMenus** parameter in the custom application's `config.xml` file (for more information, see the *TIBCO Workspace Configuration and Customization* guide).

A custom WCC application can also add custom menus and toolbar buttons to other components by calling the **getCustomMenuNode** method on the WCC application instance and providing its own handling for adding the menu or toolbar specified by the node. For an example of how this is done for the “MainAppToolbar” in the Workspace application, see the **loadWorkspaceCustomMenu** method, which is located as follows:

`StudioHome\wcc\version\JSXAPPS\workspace\application\js\AppMain.js`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.

Also see: loadCustomMenu

Syntax

```
getCustomMenuNode(type,
                  parentId);
```

Parameters

- *type* - (String) The custom type:
  - “menu” - a custom menu item.
  - “toolbar” - a custom toolbar button.
- *parentId* - (String) The value of the `config.xml` menu element parent attribute to match. See loadCustomMenu for valid values.

Returns

- jsx3.xml.Entity - The XML node, if found, otherwise, a null is returned.

registerLongLivedProcess

Registers a long-lived process. This can be used to prevent a session timeout from interrupting the completion of a long-lived server request. While a long-lived process is active, the application will periodically 'ping' the server if required to keep the session from timing out.

The cancelLongLivedProcess method should be called when the process is complete.

Syntax

```
registerLongLivedProcess(processId);
```

Parameters

- *processId* - (String) Identifies the process to register.

Returns

none

cancelLongLivedProcess

Cancels a long-lived process that was previously registered with the **registerLongLivedProcess** method.

See also: registerLongLivedProcess

Syntax

```
cancelLongLivedProcess(processId);
```

Parameters

- *processId* - (String) Identifies the process to cancel.

Returns

none

isExternalLogin

Returns the value of the **isExternalLogin** property, which indicates whether or not there is an authenticated login by an external application.

Syntax

```
isExternalLogin();
```

Parameters

none

Returns

- boolean

getLocaleKey

Returns the current Options setting for localeKey.

The localeKey is in the form of a lowercase, two-letter ISO-639 language code (ll) and optionally, the uppercase, two-letter ISO-3166 country code (CC), if the locale is country-specific.

If the country code is included, it is separated from the language code with an underscore, in the form “ll_CC” (for example, “es_MX” for Mexican Spanish).

Syntax

```
getLocaleKey();
```

Parameters

none

Returns

- String

getBaseUrl

Returns the base URL to the Action Processor.

Syntax

```
getBaseUr();
```

Parameters

none

Returns

- String

getUsername

Returns the user name of the logged-in user.

Syntax

```
getUsername();
```

Parameters

none

Returns

- String

getUserGuid

Returns the GUID of the logged-in user.

Syntax

```
getUserGuid();
```

Parameters

none

Returns

- String

getChannelId

Returns the Channel ID used to locate work item forms.

Syntax

```
getChannelId();
```

Parameters

none

Returns

- String

setChannelId

Sets the value of the Channel ID used to locate work item forms.

Syntax

```
setChannelId(value);
```

Parameters

- *value* - (String) The value to set for the Channel ID.

Returns

none

getBrowserFeatures

Returns an Object with properties that specifies features of browser windows in which work items are displayed.

For information about browser features, see the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
getBrowserFeatures();
```

Parameters

none

Returns

- Object

getSystemActionsAuthorized

Returns an Object with a property for each System Action name. The value of each property is True if the System Action is authorized; False if it is not.

An empty Object is returned if there is no logged-in user.

For information about system actions, see the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
getSystemActionsAuthorized();
```

Parameters

none

Returns

- Object

getUserAccessAuthorized

Returns an Object with a property for each user access control name. The value of each property is True if the user access is authorized; False if it is not.

For information about user access controls, see the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
getUserAccessAuthorized();
```

Parameters

none

Returns

- Object

getPrivilegeNames

Returns a comma-separated list of privileges held by the currently logged-in user.

Syntax

```
getPrivilegeNames();
```

Parameters

none

Returns

- String

loadUrlArgs

Returns an Object with a property for each URL argument for the currently loaded page. The value of each property is the value set for the argument on the URL.

Syntax

```
loadUrlArgs();
```

Parameters

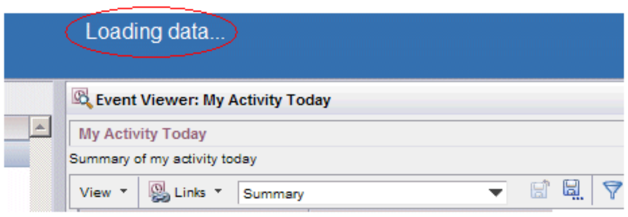
none

Returns

- Object

Data Mask Methods

The data mask is the “Loading data ...” text that is displayed while the application is retrieving information from the server and rendering it on the screen.



Also see: DataMask Component Public Methods.

Application class methods are available for working with the data mask.

isMaskActive

Returns a boolean value indicating whether or not the “Loading data...” data mask is currently displayed in the application.

Syntax

```
isMaskActive();
```

Parameters

none

Returns

- boolean - The value of the **maskActive** property: true if the data mask is currently displayed; false if it is not currently displayed.

showMask

Publishes a **showDataMask** PageBus event, and sets the **maskActive** property to true.

You can also provide an optional *msg* parameter, which, if provided, is used for the mask message.

Also see showMaskFunction to perform the same actions as this method, plus call a function asynchronously.

Syntax

```
showMask(msg);
```

Parameters

- *msg* - (String) (optional) The message to include in the data mask. If *msg* is omitted, the value of the **maskMessage** property is used, which defaults to the value in “txtDefaultMaskMessage” in the `\JSXAPPS\base\locale\locale.xml` file.

Returns

none

showMaskFunction

Publishes a **showDataMask** PageBus event, sets the **maskActive** property to true, and calls the specified function asynchronously.

The **hideDataMask** PageBus event is called when the specified function exits, unless there are other pending asynchronous calls for which showMaskFunction was called that have not yet completed.

Syntax

```
showMaskFunction(inst,
                 func,
                 argsArray,
                 msg);
```

Parameters

- *inst* - (Object) The instance on which the function is called.
- *func* - (Function) The function to call asynchronously.

- *argsArray* - (Object) An Array of function arguments.
- *msg* - (Object) (optional) The message to include in the data mask. If *msg* is omitted, the value of the **maskMessage** property is used, which defaults to the value in “txtDefaultMaskMessage” in the `\JSXAPPS\base\locale\locale.xml` file.

Returns

none

hideMask

Publishes a **hideDataMask** PageBus event, and sets the **maskActive** property to false.

If there are other pending asynchronous calls for which **showMask** was called that have not yet completed, this call has no effect.

Syntax

```
hideMask();
```

Parameters

none

Returns

none

Options Methods

User options establish default settings for each user that logs into your custom WCC application. These include things such as whether the work item list or process instance list is displayed when the user first logs in, whether or not preview is turned on by default, the size/location of work item forms, etc.

Application class methods are available for working with options.

getOptionValue

Returns the value of the specified option property.

Syntax

```
getOptionValue(optionName);
```

Parameters

- *optionName* - (String) The name of the option property. Use the **getOptionNames** method (see below) to get the available option property names.

Returns

- String - The value of the specified option property.

getOptionNames

Returns the available option property names. These names can be passed into the **getOptionValue** method (see above) to get the corresponding option property value.

Syntax

```
getOptionNames();
```

Parameters

none

Returns

- Array<Strings> - The property names used to store options data.

Application Log Methods

Application class methods are available for working with the Application Log.

appLogZeroTimer

Resets the Application Log base-line timer to an effective zero elapsed time. You can optionally append a message to the log zero-time message.

Syntax

```
appLogZeroTimer(msg);
```

Parameters

- *msg* - (String) (optional) The message to append to the zero-time message.

Returns

none

log

Adds a new Application Log entry at the INFO level. (This is a convenience alias for **logInfo**.)

Also see: logInfo

Syntax

```
log(msg);
```

Parameters

- *msg* - (String) The message to add to the Application Log.

Returns

none

logDebug

Adds a new Application Log entry at the DEBUG level.

Syntax

```
logDebug(msg);
```

Parameters

- *msg* - (String) The Debug message to add to the Application Log.

Returns

none

logTrace

Adds a new Application Log entry at the TRACE level.

Syntax

```
logTrace(msg);
```

Parameters

- *msg* - (String) The Trace message to add to the Application Log.

Returns

none

logFatal

Adds an Application Log entry at the FATAL level.

Syntax

```
logFatal(msg);
```

Parameters

- *msg* - (String) The Fatal message to add to the Application Log.

Returns

none

logError

Adds an Application Log entry at the ERROR level.

Syntax

```
logError(msg);
```

Parameters

- *msg* - (String) The Error message to add to the Application Log.

Returns

none

logWarn

Adds an Application Log entry at the WARN level.

Syntax

```
logWarn(msg);
```

Parameters

- *msg* - (String) The message to add to the Application Log.

Returns

none

logInfo

Adds an Application Log entry at the INFO level.

Also see log .

Syntax

```
logInfo(msg);
```

Parameters

- *msg* - (String) The message to add to the Application Log.

Returns

none

getAppLogLevel

Returns an integer representing the current log level.

The levels are:

- 0 - OFF
- 1 - FATAL
- 2 - ERROR
- 3 - WARN
- 4 - INFO
- 5 - DEBUG
- 6 - TRACE

Syntax

```
getAppLogLevel();
```

Parameters

none

Returns

- integer

getAppLogLevelName

Returns the current log level name.

The log level names are:

- "OFF"
- "FATAL"
- "ERROR"
- "WARN"
- "INFO"
- "DEBUG"
- "TRACE"

Syntax

```
getAppLogLevelName();
```

Parameters

none

Returns

- String

setAppLogLevel

Sets the log level, using an integer value.

This can be set for the:

- application instance,
- **jsx3.util.Logger** used by this application instance,
- **jsx3.util.Logger** used by the static com.tibco.wcc.base.log(), and
- **jsx3.util.Logger.GLOBAL** logger instance.

Syntax

```
setAppLogLevel(value);
```

Parameters

- *value* - (integer) Specifies the log level, as follows:
  - 0 - OFF
  - 1 - FATAL
  - 2 - ERROR
  - 3 - WARN
  - 4 - INFO
  - 5 - DEBUG
  - 6 - TRACE

Returns

none

setAppLogLevelName

Sets the log level, using a name.

This can be set for the:

- application instance,
- **jsx3.util.Logger** used by this application instance,
- **jsx3.util.Logger** used by the static com.tibco.wcc.base.log(), and
- **jsx3.util.Logger.GLOBAL** logger instance.

Syntax

```
setAppLogLevelName(value);
```

Parameters

- *value* - (String) Specifies the log level, as follows:
  - “OFF”
  - “FATAL”
  - “ERROR”
  - “WARN”
  - “INFO”
  - “DEBUG”
  - “TRACE”

Returns

none

isEchoToJsxLog

Returns True if Application Log entries are echoed to the Application Monitor.

For more information, see the *Workspace Logs* appendix in the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
isEchoToJsxLog();
```

Parameters

none

Returns

- boolean

setEchoToJsxLog

Specifies whether Application Log entries are echoed to the Application Monitor.

For more information, see the *Workspace Logs* appendix in the *TIBCO Workspace Configuration and Customization* guide.

Syntax

```
setEchoToJsxLog(value);
```

Parameters

- *value* - (boolean) True = echo Application Log entries to the Application Monitor; False = entries are not echoed.

Returns

none

getJsxLogger

Returns the logger used to write messages to the Application Monitor if **echoToJsxLog** is True.

Also see isEchoToJsxLog.

Syntax

```
getJsxLogger();
```

Parameters

none

Returns

- jsx3.util.Logger

logException

Adds a new Application Log entry for the specified message and exception object at the ERROR level and returns the resulting message string.

Syntax

```
logException(msg, exception);
```

Parameters

- *msg* - (String) Message to add to log entry, in the format:
  - “Error message: %eMsg% more message text”

    where %eMsg% is an optional token that is replaced with the message of the *exception* parameter. If %eMsg% is omitted, the exception message is added to the *msg* parameter string.

    If **exception.getMessage()** is defined, this exception message is used. Else if **exception.message** is defined, this exception message is used. Otherwise (or if exception is omitted), the exception message used is: “Unknown Exception Type”
- *exception* - (Object) An exception object.

Returns

- String - The exception message written to the Application Log.

logAlways

Always writes message to both the Application Log and **jsx3.log** at the INFO level, regardless of what is configured for **appLogLevel** and **echoToJsxLog**.

Syntax

```
logAlways(msg);
```

Parameters

- *msg* - (String) The message to add to the Application Log and **jsx3.log**.

Returns

none

logStackTrace

Adds a new Application Log entry and a stackTrace to both the Application Log and **jsx3.log**.

Syntax

```
logStackTrace(msg);
```

Parameters

- *msg* - (String) The message to add to the Application Log and **jsx3.log**.

Returns

none

logHierarchy

Adds a new Application Log entry for the hierarchy of the given object to both the Application Log and **jsx3.log**.

Syntax

```
logHierarchy(obj);
```

Parameters

- *obj* (jsx3.app.Model) The GUI object to show a hierarchy for.

Returns

none

logTree

Adds a new Application Log entry for a tree view of children of the given object to both the Application Log and **jsx3.log**.

Syntax

```
logTree(obj);
```

Parameters

- *obj* - (jsx3.app.Model) The GUI object to show a tree for.

Returns

none

logStart

Adds the given message as a timed Application Log entry. This will be flagged as a timed entry and a call to **logStop** with the matching *msg* will set the elapsed time from the **logStart** call to the **logStop** call. This time is logged under the timer column.

Note that the elapsed time from this **logStart** call to the following **log** call will also be recorded. This appears under the elapsed column.

Syntax

```
logStart(msg, forceLog);
```

Parameters

- *msg* (String) The message to be added to the log.
- *forceLog* (boolean) (Optional) If true, log is written even when **appLogLevel** is less than "INFO".

Returns

none

logStop

Sets the elapsed time for a timed Application Log entry with the matching *msg*. The *appendMsg* argument, if not null, is added to the *msg* in the log entry. If a matching **logStart** *msg* is not found, a new log is added, which is pre-pended with a '- ' character.

Syntax

```
logStop(msg, appendMsg, forceLog);
```

Parameters

- *msg* (String) The message to be added to the log.
- *appendMsg* (String) (Optional) A message to append to *msg*.
- *forceLog* (boolean) (Optional) If true, log message is written even when **appLogLevel** is less than "INFO".

Returns

none

getAppLogText

Returns the value of the Application Log text.

Syntax

```
getAppLogText();
```

Parameters

none

Returns

none

clearAppLog

Clears all except the original initialization entries, then resets the Application Log baseline time to an effective zero elapsed time.

Syntax

```
clearAppLog();
```

Parameters

none

Returns

none

User Data Methods

Application class methods are available for working with user data.

getUserDataAttributes

Reads attribute values from the document at UserData *settingId* that match property names in the Object specified by *objAttributes*, then sets the value of the *objAttributes* properties with the values in the persisted UserData document.

If the attribute does not exist in the document, the value of the *objAttributes* property is not changed.

Syntax

```
getUserDataAttributes(settingId,
                      objAttributes);
```

Parameters

- *settingId* - (String) The name used for the persisted UserData.
- *objAttributes* - (Object) Object whose properties are set with the values from the persisted UserData document.

Returns

none

setUserDataAttributes

Sets persisted user data using the values in the properties of the specified Object. If the property value in the Object is null, it is ignored and no attribute value is persisted.

Syntax

```
setUserDataAttributes(SettingId,
                      objAttributes);
```

Parameters

- *settingId* - (String) The name used for the persisted UserData whose attributes are set from the specified Object properties.
- *objAttributes* - (Object) Object whose property values are written to the persisted UserData document.

Returns

none

User Message Methods

Application class methods are available for working with user messages.

showUserMessage

Displays the given *msg* in an alert dialog. This should be called for all messages to be displayed to the application user rather than calling **alert** directly.

The current implementation uses **alert**, which is blocking. If this is changed, this behavior needs to be considered for all calls that might depend on a blocking call.

Syntax

```
showUserMessage(msg, windowContext);
```

Parameters

- *msg* - (String) The string to display in the alert dialog.
- *windowContext* - (window) (Optional) The browser window instance. This can be passed in when called from a child browser window to prevent focus moving to the parent browser window.

Returns

none

showErrorMessage

Displays the given *errorMsg* in a dialog. The dialog displays the *errorMsg* parameter as the primary error message and provides a “Show Details” button which expands the dialog to display the *errorDetails* parameter.

This call is not blocking and returns before the user responds to dialog.

Syntax

```
showErrorMessage(errorMsg, errorDetails);
```

Parameters

- *errorMsg* (String) The string to display as the primary error message.
- *errorDetails* (String) If this string is not null and not empty, a “Show Details” button is displayed, that when clicked, expands to show error details (exception/stack trace) in a scrollable text area.

Returns

none

confirmUserMessage

Displays the given *msg* in a confirm dialog. This should be called for all messages to be confirmed to the application user rather than calling **confirm** directly. Note that future enhancements or extensions of this method may be applied to change how messages are displayed to the user, logged, or otherwise processed.

The current implementation uses **confirm**, which is blocking. If this is changed, this behavior needs to be considered for all calls that might depend on a blocking call.

Syntax

```
confirmUserMessage(msg, windowContext);
```

Parameters

- *msg* (String) The string to display in the confirm dialog.
- *windowContext* (window) (Optional) The browser window instance. This can be passed in when called from a child browser window to prevent focus moving to the parent browser window.

Returns

- (boolean) Returns true if user selects **OK** from the ensuing dialog; False if the user selects **Cancel**.

Display Caption Methods

Application class methods are available for working with captions in your custom WCC application.

setCaptionBar

Displays an image and text on the specified caption bar.

This also sets the image and text on the taskbar if one is displayed.

Syntax

```
setCaptionBar(bar,
              img,
              text);
```

Parameters

- *bar* - (jsx3.gui.WindowBar or jsx3.gui.Block) The caption bar.
- *img* - (String) An image string in the form: '<img scr="path"/>'
- *text* - (String) The text to show in the caption.

Returns

none

setCaptionBarText

Sets the display caption on the specified bar.

This also sets the image and text on a taskbar button if:

- the bar type is jsx3.gui.WindowBar
- the *setTaskButton* parameter is true

- a taskbar button exists

Syntax

```
setCaptionBarText(bar,
                  textHtml,
                  setTaskButton);
```

Parameters

- *bar* - (jsx3.gui.WindowBar or jsx3.gui.Block) The caption bar.
- *textHtml* - (String) A text or HTML string value to set on the caption bar.
- *setTaskButton* - (boolean) Pass true if *textHtml* should be applied to setting the taskbar button. This only applies if caption bar is of type jsx3.gui.WindowBar. If this is set to true, *textHtml* must be of the form:
  - `<img src='path/?.gif'><span style='styles'>Caption String</span>`

Returns

none

Exception Handling Methods

Application class methods are used to get or set the **toolsErrorHandlingEnabled** property.

setToolsErrorHandlingEnabled

Sets the value of the **toolsErrorHandlingEnabled** property, which determines how to handle exceptions returned by the Action Processor.

Exceptions can be returned by the Action Processor when you call any of the *Tools interface* methods (see Tools Interface Methods).

Also see Exception Handling.

Syntax

```
setToolsErrorHandlingEnabled(value);
```

Parameters

- *value* - (boolean):
  - False (the default) = the exception will be passed to the caller, which means your custom application should handle any exceptions that are returned from the Action Processor.
  - True = the exception will be handled for you, that is, a message is displayed to the user, that is, the exception is not passed to the caller. (Note - The **ShowErrorDetails** user access control determines how much detail is displayed to the user for an Action Processor exception.)

Returns

none

isToolsErrorHandlingEnabled

Returns the value of the **toolsErrorHandlingEnabled** property.

Also see setToolsErrorHandlingEnabled.

Syntax

```
isToolsErrorHandlingEnabled();
```

Parameters

none

Returns

- boolean

Tools Interface Methods

The Application class methods described in the following sub-topics are considered the *Tools Interface* methods. These methods differ from the other Application class methods in that they (the Tools Interface methods) make calls to the Action Processor, and can therefore, receive exceptions back from the Action Processor.

These exceptions from the Action Processor can be handled as described in Exception Handling.

Also note that a number of the Tools Interface methods (i.e., those pertaining to work items, business services, and process instances) are overloaded and have two signatures: one for a single work item or process instance, and another for multiple work items and process instances.

The Tools Interface methods are divided into the following areas:

- Access Methods
- Work Item Methods
- Process Instance Methods
- Business Service Methods
- View Methods
- User Options Methods
- Organization Browser Methods
- Configuration Methods
- Locale Methods

Exception Handling

There are two levels of exceptions that can be returned when calling the Tools Interface methods: parameter exceptions and Action Processor exceptions.

- **Parameter exception** - This type of exception is returned in the following situations:
  - One or more of the parameters passed in the method call is undefined.
  - If a *parent* parameter is passed, it is defined, but does not implement the General Interface **setChild** method.

    Parameter exceptions are always returned to the calling application.
- **Action Processor exception** - This type of exception results when the Action Processor returns an error.

  There is a property, **toolsErrorHandlingEnabled**, whose value determines how Action Processor exceptions are handled.

  - If **toolsErrorHandlingEnabled** = false (the default), your application should handle any exceptions that are returned.

- If **toolsErrorHandlingEnabled** = true, the exception will be handled for you, that is, a message is displayed to the user when an Action Processor exception is returned. (Note - The **ShowErrorDetails** user access control determines how much detail is displayed to the user for an Action Processor exception.)

  Two Application class methods are available that can be used to specify how exceptions returned by the Action Processor are handled:

- **setToolsErrorHandlingEnabled** - Allows you to set the **toolsErrorHandlingEnabled** property. For more information, see setToolsErrorHandlingEnabled.
- **isToolsErrorHandlingEnabled** - Returns the current value of the **toolsErrorHandlingEnabled** property. For more information, see isToolsErrorHandlingEnabled.

Running the Sample Application

A sample application is provided that gives an example of the available Tools Interface methods in the Application class. It provides a UI that allows you to enter the needed information (e.g., work item ID, version number, etc.), then execute a method.

The sample application is located in the following directory:

```
StudioHome\wcc\version\samples\wccApplicationMethods\
```

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.

Also note that a `wccApplicationMethods.create.war.cmd` file is provided that can be used to create a WAR file that can be used to deploy the sample application to a runtime node. For more information, see the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Configuration and Customization* guide.

Procedure

1. Copy the following directory:
   ```
   \samples\wccApplicationMethods\JSXAPPS\wccApplicationMethods\
   ```
2. Paste the directory you've just copied into the following directory:

   `StudioHome\wcc\version\JSXAPPS\`
3. Open the following file with an editor:

   `StudioHome\wcc\version\JSXAPPS\wccApplicationMethods\config.xml`
4. Locate the **ActionProcessors** record in the `config.xml` file and modify the **baseUrl** attribute to point to your Action Processor:
   ```
   <record jsxid="ActionProcessors" type="workspace">
       <ActionProcessor
          weighting="100"
          baseUrl="http://Austin:8080/bpm/actionprocessor/actionprocessor.servlet">
       </ActionProcessor>
   </record>
   ```
   The **baseUrl** attribute string must be in the form:

   `http://Host:Port/bpm/actionprocessor/actionProcessor.servlet`

   where:

   - *Host* is the name of the machine hosting the BPM runtime.

- *Port* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

  For more information about the **ActionProcessors** record, see the *TIBCO Workspace Configuration and Customization* guide.

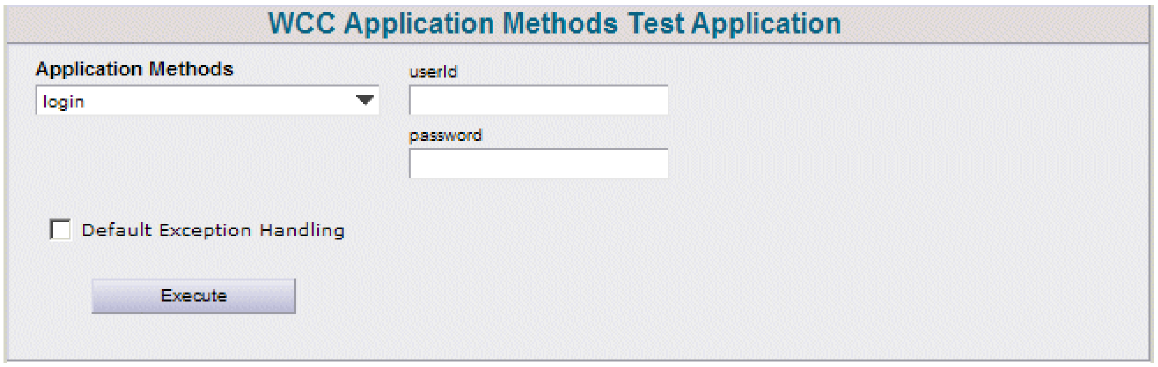
5. Save and close the `config.xml` file.
6. Copy the `wccApplicationMethods.html` file from the following directory:

   ```
   StudioHome\wcc\version\Samples\wccApplicationMethods\
   ```
7. Paste the file you've just copied into the installation directory:

   `StudioHome\wcc\version\`
8. Ensure your server is running.
9. Start the sample application by pointing your browser to the following:

   `StudioHome\wcc\version\wccApplicationMethods.html`

   The following screen is displayed:

   
   WCC Application Methods Test Application
   Application Methods
   login
   userId
   password
   Default Exception Handling
   Execute

   The **Application Methods** field provides a drop down list containing all of the available application methods. When you select a method from the list, the fields on the screen change to the appropriate fields for the selected method.
10. Enter any required information, then click the **Execute** button to execute the method.

    Note that the **Default Exception Handling** check box allows you to specify how exception handling will be accomplished for methods called from the sample application. This check box sets the **toolsErrorHandlingEnabled** property. Checking the box sets the property to true, which causes a message to be displayed to the user if an Action Processor exception is returned. Unchecking the box sets the property to false, which disables the default error handling; messages are not displayed to the user if an Action Processor exception is returned — the exception should be handled by the calling application.

Access Methods

Tools Interface methods are available for accessing your custom WCC application.

login

This method issues a login request for the specified user.

Note that this method has no affect if the user is already logged in.

Syntax

```
login(userId,
      password);
```

Parameters

- *userId* - (String) The name of the user to log in.
- *password* - (String) The password for the user logged in.

logout

This method issues a logout request for the current user.

Syntax

```
logout();
```

Parameters

none

Returns

none

initializeExternalLogin

This method initializes the application after login is completed by an external application.

This method must be called if the login is done by an external application. However, it must *not* be called if the login is done by the current application (i.e., by the **wcc.Login** component or the **app.login()** method, as these call the **initializeExternalLogin** method).

Note that calling this method has no affect if the user is already logged in.

Syntax

```
initializeExternalLogin();
```

Parameters

none

Returns

none

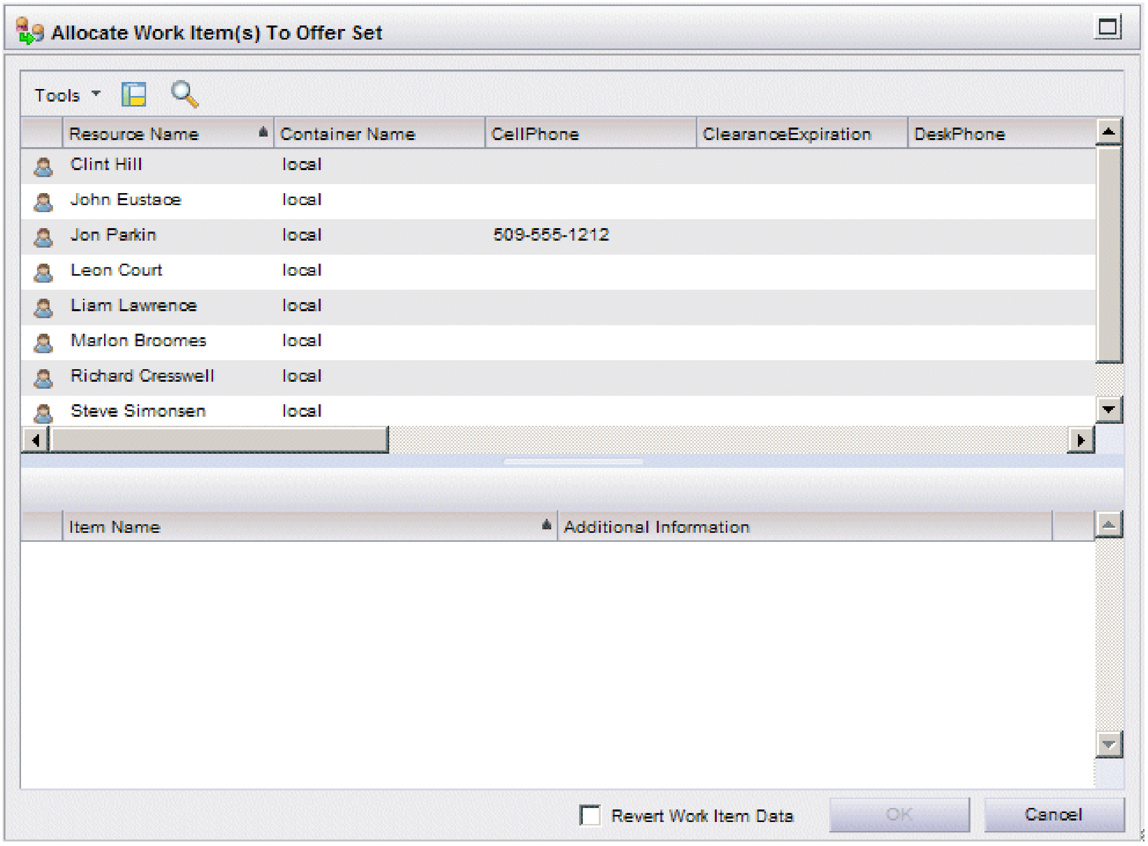
Work Item Methods

Tools Interface methods are available for working with work items in your custom WCC application.

allocateWorkItemToOfferSet

This function allows a user to allocate one or more work items to a user from the original offer set, that is, the group of users to which the work items were originally offered. It removes the work items from the user's work item list, then adds them to the Inbox of the user to whom they have been allocated, with a state of Allocated.

Executing this method causes a dialog similar to the following to be displayed, either in the parent block, or in a dialog, depending on the value passed in the *parent* parameter:



Resource Name	Container Name	CellPhone	ClearanceExpiration	DeskPhone
Clint Hill	local			
John Eustace	local			
Jon Parkin	local	509-555-1212		
Leon Court	local			
Liam Lawrence	local			
Marlon Broomes	local			
Richard Cresswell	local			
Steve Simonsen	local			

Item Name	Additional Information

This method has two signatures: one to allocate a single work item, and one to allocate multiple work items.

This component publishes the **closeAssignWorkItems** event. The application should subscribe to this event to detect when user has clicked either the **OK** or **Cancel** button. At that time, assignments, if any, will have been completed.

Ideally, the component will be displayed modally. If you choose to display it non-modally and allow the same work item to appear in multiple instances, keep in mind that only the first of those to perform an assignment of that work item will succeed. Other instances will get an exception.

Syntax

```
allocateWorkItemToOfferSet(id,
                           version,
                           state,
                           workType,
                           parent);
```

```
allocateWorkItemToOfferSet(workItems,
                           workType,
                           parent);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to allocate to the original offer set.
- *version* - (integer) Specifies the version number of the work item you want to allocate. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *state* - (String) Identifies the state of the work item you want to allocate. The possible states are:

  - OFFERED
  - ALLOCATED
  - PENDED
- *workType* - (constant) Identifies whether the work item(s) being allocated belong to the current user's work set or are from a work set the current user supervises. The possible values are:
  - com.tibco.wcc.components.WorkViews.TYPE_MYWORK
  - com.tibco.wcc.components.WorkViews.TYPE_SUPERVISEDWORK
- *parent* - (Object) (optional) Identifies the General Interface component where the interface is to be loaded. If omitted, the interface is loaded in a dialog.
- *workItems* - (Array<Objects>) These identify the work items to be allocated. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)
  - state (String)

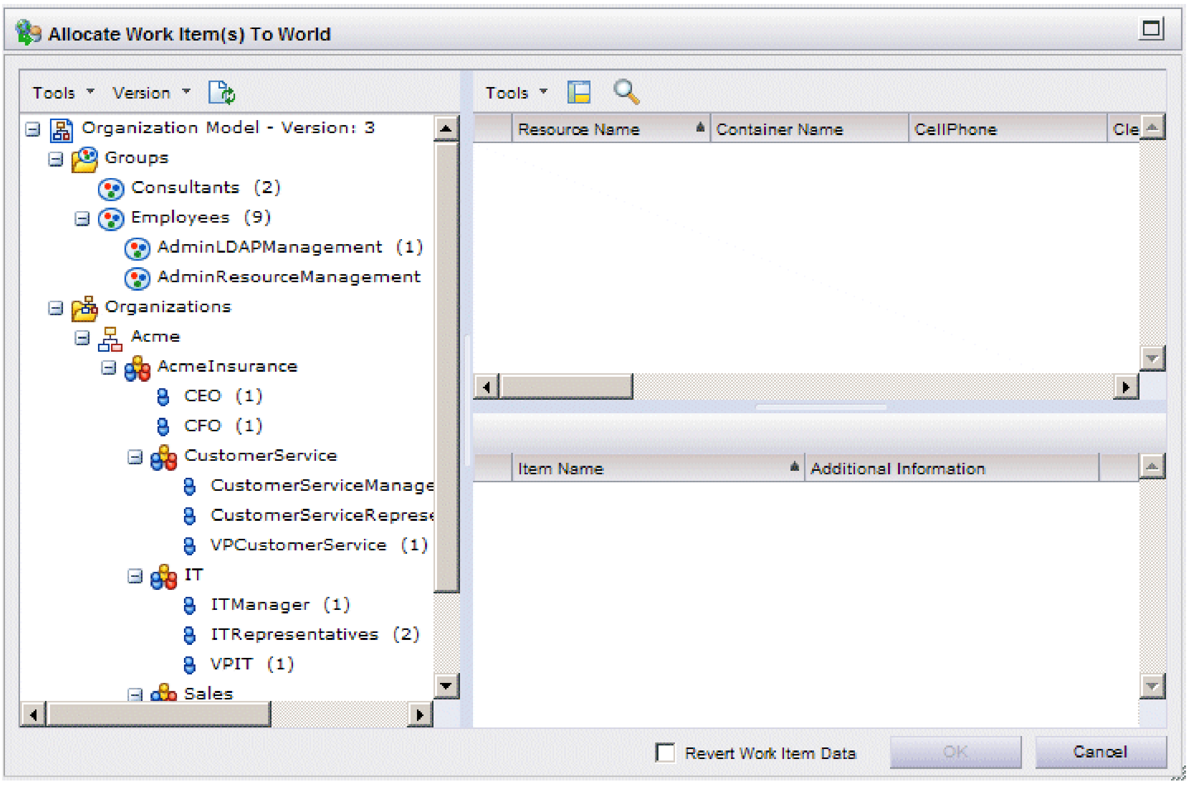
Returns

none

allocateWorkItemToWorld

This function allows the user to allocate one or more work items to any available user. The allocated work item is removed from the current user's work item list, and appears in the work item list of the recipient, with a state of Allocated.

Executing this method causes a dialog similar to the following to be displayed, either in the parent block, or in a dialog, depending on the value passed in the *parent* parameter:



This method has two signatures: one to allocate a single work item, and one to allocate multiple work items.

This component publishes the **closeAssignWorkItems** event. The application should subscribe to this event to detect when user has clicked either the **OK** or **Cancel** button. At that time, assignments, if any, will have been completed.

Ideally, the component will be displayed modally. If you choose to display it non-modally and allow the same work item to appear in multiple instances, keep in mind that only the first of those to perform an assignment of that work item will succeed. Other instances will get an exception.

Syntax

```
allocateWorkItemToWorld(id,
                        version,
                        state,
                        workType,
                        parent);
```

```
allocateWorkItemToWorld(workItems,
                        workType,
                        parent);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to allocate to another user.
- *version* - (integer) Specifies the version number of the work item you want to allocate. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *state* - (String) Identifies the state of the work item you want to allocate. The possible states are:
  - ALLOCATED
  - PENDED
- *workType* - (constant) Identifies whether the work item(s) being allocated belong to the current user's work set or are from a work set the current user supervises. The possible values are:
  - com.tibco.wcc.components.WorkViews.TYPE_MYWORK
  - com.tibco.wcc.components.WorkViews.TYPE_SUPERVISEDWORK
- *parent* - (Object) (optional) Identifies the General Interface component where the interface is to be loaded. If omitted, the interface is loaded in a dialog.
- *workItems* - (Array<Objects>) These identify the work items to be allocated. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)
  - state (String)

Returns

none

allocateWorkItemToSelf

This method allocates a work item to the current user.

The work item being allocated must have a current state of Offered. After allocation, the work item's state changes to Allocated in the current user's work item list, and the work item is removed from the work item list of all other users to whom it was being offered.

This method has two signatures: one to allocate a single work item, and one to allocate multiple work items.

Syntax

```
allocateWorkItemToSelf(id,
                       version);
```

```
allocateWorkItemToSelf(workItems);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to allocate to the current user.
- *version* - (integer) Specifies the version number of the work item you want to allocate. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *workItems* - (Array<Objects>) These identify the work items to be allocated. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

cancelWorkItem

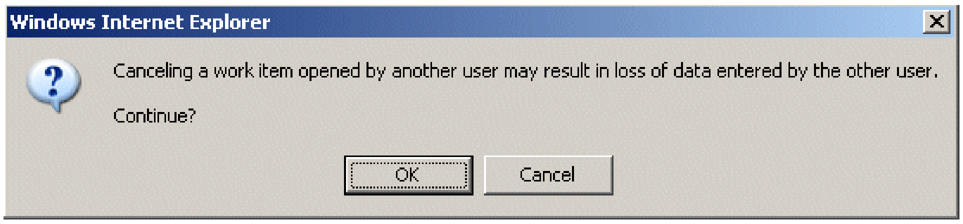
This method closes the specified work item, discarding any changes that had been made on the work item form. To cancel a work item, its current state must be Opened. When the cancel function is complete, the work item's state will be returned to the state it was in when the work item was opened.

The intention of the cancel function is for an administrator-type user to be able to forcibly close a work item left open by another user. This will cause any data entered or changed by the user who opened the work item to be lost.

This method can be used only if the work item had been opened in a floating window — it does not work if the work item is open in the Preview Pane.

This method has two signatures: one to cancel a single work item, and one to cancel multiple work items.

If you do not suppress the confirmation message, the following is displayed, warning you about the possible loss of data:


Windows Internet Explorer

Canceling a work item opened by another user may result in loss of data entered by the other user.

Continue?

Also note that the work item form that was opened by the user who originally opened the work item is not closed by the cancel function. The form must be manually closed at the workstation from which it was opened.

Syntax

```
cancelWorkItem(id,
               version,
               suppressConfirm);
```

```
cancelWorkItem(workItems,
               suppressConfirm);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to cancel.
- *version* - (integer) Specifies the version number of the work item you want to cancel. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.
- *workItems* - (Array<Objects>) These identify the work items to be cancelled. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

openNextWorkItem

This method causes the first available work item in the work item list to be opened, where “available” means a work item that is not locked nor suspended.

Also see:

Pageflow definitions can include branching that would require the simultaneous handling of multiple user tasks, each requiring the display of a form (this same scenario could also occur in embedded sub-processes set up for chained execution).

Workspace cannot currently display multiple forms resulting from these *parallel pageflows*. If one is encountered in a WCC application (including the Workspace application), a warning message is displayed. In some situations, none of the forms can be displayed and execution of the pageflow (or chaining in the sub-process) cannot proceed. If a parallel pageflow is encountered inside of a business service, it may be able to arbitrarily display one of the forms, but an exception may occur later when the form is submitted.

openNextWorkItemEx.

Syntax

```
openNextWorkItem(parent);
```

```
openNextWorkItem(channelId,
                 parent);
```

The method signature with the *channelId* parameter is deprecated as of version 1.1.0 of Workspace. New code should use the signature with only the *parent* parameter.

Parameters

- *channelId* - (String) (optional) Identifies the work presentation (forms) that will be used for the work item that is opened. If omitted, the channel ID specified in the *channelId* parameter in the application's `config.xml` file is used.
- *parent* - (Object) (optional) Identifies the General Interface component where the work item form is to be loaded. If omitted, it is loaded in a dialog.

Returns

none

openNextWorkItemEx

This method causes the next available work item in the work item list to be opened, based on specified filter and sort parameters. "Available" means a work item that is not locked nor suspended.

Pageflow definitions can include branching that would require the simultaneous handling of multiple user tasks, each requiring the display of a form (this same scenario could also occur in embedded sub-processes set up for chained execution).

Multiple forms resulting from these *parallel pageflows* cannot currently be displayed. If one is encountered in a WCC application (including the Workspace application), a warning message is displayed. In some situations, none of the forms can be displayed and execution of the pageflow (or chaining in the sub-process) cannot proceed. If a parallel pageflow is encountered inside of a business service, the application may be able to arbitrarily display one of the forms, but an exception may occur later when the form is submitted.

Syntax

```
openNextWorkItemEx(parent, filter, sort);
```

Parameters

- *parent* - (Object) (optional) Identifies the General Interface component where the work item form is to be loaded (the component must implement the General Interface **setChild** method). If omitted, the work item is loaded in a dialog.
- *filter* - (String) (optional) A filter expression to apply to the work item list prior to opening the next available work item. For information about filter syntax, see the "Sorting and Filtering Work Item Lists" topic in the *TIBCO ActiveMatrix BPM Developer's Guide*.
- *sort* - (String) (optional) The sort order to apply to the work item list prior to opening the next available work item. Specify the sort field name, following by either ASC or DESC, for ascending or descending, respectively. Separate multiple sort fields with commas. For example:

  ```
  priority ASC, startDate DESC
  ```

  Use the following sort field names in the sort parameter:

Attribute	Sort Field Name
Attribute #1 - Attribute #40	attribute1 - attribute40
Distribution Strategy	distributionStrategy
Target Date	endDate

Attribute	Sort Field Name
Instance Description	appInstanceDescription
ID	appInstance
Priority	priority
Processs Template	appName
Start Date	startDate
Work Item ID	id

Returns

none

openWorkItem

This method opens the specified work item.

This method has two signatures: one to open a single work item, and one to open multiple work items.

Pageflow definitions can include branching that would require the simultaneous handling of multiple user tasks, each requiring the display of a form (this same scenario could also occur in embedded sub-processes set up for chained execution).

Workspace cannot currently display multiple forms resulting from these *parallel pageflows*. If one is encountered in a WCC application (including the Workspace application), a warning message is displayed. In some situations, none of the forms can be displayed and execution of the pageflow (or chaining in the sub-process) cannot proceed. If a parallel pageflow is encountered inside of a business service, it may be able to arbitrarily display one of the forms, but an exception may occur later when the form is submitted.

Syntax

```
openWorkItem(id,
             version,
             parent);
```

```
openWorkItem(workItems,
             parent);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to open.
- *version* - (integer) Specifies the version number of the work item you want to open. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *parent* - (Object) (optional) Identifies the General Interface component where the work item form is to be loaded. If omitted, it is loaded in a dialog.
- *workItems* - (Array<Objects>) These identify the work items to be opened. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

reofferWorkItem

This method re-offers the specified work item.

To be re-offered, a work item must currently have a state of Allocated or Pended.

After being re-offered the work item appears in the Inbox of the user(s) to whom it was originally offered.

Note that if the work item's state is Pended, and data had been added/changed on the form when it was open, that data is retained in the work item when it is re-offered.

This method has two signatures: one to re-offer a single work item, and one to re-offer multiple work items.

Syntax

```
reofferWorkItem(id,
                version);
```

```
reofferWorkItem(workItems);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to re-offer.
- *version* - (integer) Specifies the version number of the work item you want to re-offer. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *workItems* - (Array<Objects>) These identify the work items to be opened. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

skipWorkItem

This method marks the specified work item as complete, removes it from the work item list, and causes the process to advance. It has the same affect as opening the work item and submitting it.

To skip a work item, the work item cannot have any required data fields that are not filled in.

This method has two signatures: one to skip a single work item, and one to skip multiple work items.

Syntax

```
skipWorkItem(id,
             version);
```

```
skipWorkItem(workItems);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to skip.

- *version* - (integer) Specifies the version number of the work item you want to skip. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *workItems* - (Array<Objects>) These identify the work items to be skipped. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

pendWorkItem

This method causes the specified work item(s) to be hidden in the work item list until a specified date/ time, or period of time has expired. The work item becomes visible again when the date/time occurs, or period of time expires.

Work items can be pended only if they have a state of Allocated.

This method has two signatures: one to pend a single work item, and one to pend multiple work items.

Syntax

```
pendWorkItem(id,
             version,
             date,
             duration);
```

```
pendWorkItem(workItems,
             date,
             duration);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item to pend.
- *version* - (integer) Specifies the version number of the work item you want to pend. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- date - (String) Date, in UTC format, when pending until a specific date.
- duration - (Object) An Object containing the following properties — used when pending for a period of time:
  - years
  - months
  - weeks
  - days
  - hours
  - minutes
- *workItems* - (Array<Objects>) These identify the work items to be pended. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

prioritizeWorkItem

This method alters the priority of the specified work item(s).

There are two system actions controlling a user's right to change work item priority:

- **BRM.changeAllocatedWorkItemPriority** - Users with this system action can alter the priority on allocated and pended work items.
- **BRM.changeAnyWorkItemPriority** - Users with this system action can alter the priority on allocated, pended, and offered work items.

This method has two signatures: one to set the priority for a single work item, and another for multiple work items.

Syntax

```
prioritizeWorkItem(id,
                   version,
                   priority);
```

```
prioritizeWorkItem(workItems,
                   priority);
```

Parameters

- *id* - (integer) A work item ID, identifying the work item.
- *version* - (integer) Specifies the version number of the work item whose priority is being set. The version number indicates how many times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by one each time it changes state.
- *priority* - (integer) A number, from 0 - 100, indicating the new priority.
- *workItems* - (Array<Objects>) These identify the work items whose priority is being set. Each Object in the array must have the following properties, identifying an existing work item (see above for definitions):
  - workItemId (integer)
  - workItemVersion (integer)

Returns

none

Process Instance Methods

Tools Interface methods are available for working with process instances in your custom WCC application.

startProcessInstance

This method starts an instance of the specified process template.

Note that process instances are typically started via a business service, rather than independently — see startBusinessService.

Syntax

```
startProcessInstance(name,
                     module,
                     version);
```

Parameters

- *name* - (String) The name of the process template for which an instance is to be started.
- *module* - (String) The module of the process template to start.
- *version* - (String) (optional) The version of the process template to start.

Returns

none

cancelProcessInstance

This method cancels the specified process instance. This results in all work items associated with that process instance also being cancelled — they are removed from work lists of all users to whom they were offered or allocated.

This method has two signatures: one to cancel a single process instance, and one to cancel multiple process instances.

Syntax

```
cancelProcessInstance(id,
                      suppressConfirm);
```

```
cancelProcessInstance(ids,
                      suppressConfirm);
```

Parameters

- *id* - (String) A process instance ID, identifying the process instance to cancel.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.
- *id*s - (String[]) An array of process instances IDs, identifying the process instances to cancel.

Returns

none

resumeProcessInstance

This method resumes the specified suspended process instance.

The specified process instance’s status changes from SUSPENDED to ACTIVE. And the associated work items that were suspended when the process instance was suspended are also resumed — their state is changed from SUSPENDED to the state they were in when the process instance was suspended.

This method has two signatures: one to resume a single process instance, and one to resume multiple process instances.

Also see: suspendProcessInstance.

Syntax

```
resumeProcessInstance(id,
                      suppressConfirm);
```

```
resumeProcessInstance(ids,
                      suppressConfirm);
```

Parameters

- *id* - (String) A process instance ID, identifying the process instance to resume.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.
- *id*s - (String[]) An array of process instances IDs, identifying the process instances to resume.

Returns

none

suspendProcessInstance

This method suspends the specified process instance.

This causes the process instance's status to change to Suspended. It also results in all work items associated with the process instance to also become suspended; when suspended, they cannot be opened.

Note that work items that have a state of Suspended are *hidden* in the work item list. The **View** menu on the work item list provides selections that allow the user to display visible and/or hidden work items.

This method has two signatures: one to suspend a single process instance, and one to suspend multiple process instances.

Also see: resumeProcessInstance.

Syntax

```
suspendProcessInstance(id,
                       suppressConfirm);
```

```
suspendProcessInstance(ids,
                       suppressConfirm);
```

Parameters

- *id* - (String) A process instance ID, identifying the process instance to suspend.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.
- *id*s - (String[]) An array of process instances IDs, identifying the process instances to suspend.

Returns

none

resumeHaltedProcessInstance

This method resumes the halted process instance so that the instance itself goes into a failed state because of the failed task. This would typically be used if the reason for the failed task cannot be resolved.

This method has two signatures: one for a single halted process instance, and one for multiple halted process instances.

For more information, see the "Halted Process Instances" topic in the *TIBCO Workspace User's Guide*.

Syntax

```
resumeHaltedProcessInstance(id,
                            suppressConfirm);
```

```
resumeHaltedProcessInstance(ids,
                            suppressConfirm);
```

Parameters

- *id* - (String) A process instance ID, identifying the halted process instance to resume.
- *id*s - (String[]) An array of process instances IDs, identifying the halted process instances to resume.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.

Returns

none

retryProcessInstance

This method causes the previously failed task to be retried so that the halted process instance can progress normally. This method is used if the reason for the failed task has been resolved, for example, if it failed because a database connection had been lost, but has now been restored.

This method has two signatures: one for a single halted process instance, and one for multiple halted process instances.

For more information, see the "Halted Process Instances" topic in the *TIBCO Workspace User's Guide*.

Syntax

```
retryProcessInstance(id,
                     suppressConfirm);
```

```
retryProcessInstance(ids,
                     suppressConfirm);
```

Parameters

- *id* - (String) A process instance ID, identifying the halted process instance to retry.
- *id*s - (String[]) An array of process instances IDs, identifying the halted process instances to retry.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.

Returns

none

ignoreProcessInstance

This method causes the failed task to be skipped in the halted process instance. The process instance continues processing from the point in the process after the failed task.

This method has two signatures: one for a single halted process instance, and one for multiple halted process instances.

For more information, see the "Halted Process Instances" topic in the *TIBCO Workspace User's Guide*.

Syntax

```
ignoreProcessInstance(id,
                      suppressConfirm);
```

```
ignoreProcessInstance(ids,
                      suppressConfirm);
```

Parameters

- *id* - (String) A process instance ID, identifying the halted process instance in which the failed task is to be ignored.
- *id*s - (String[]) An array of process instances IDs, identifying the halted process instances in which the failed task is to be ignored.
- *suppressConfirm* - (boolean) (optional) If true, suppresses the confirmation message. If false (the default), the confirmation message is displayed.

Returns

none

Business Service Methods

Tools Interface method is available for working with business services in your custom WCC application.

startBusinessService

This method starts a business service.

This method has two signatures: one to start a single business service, and one to start multiple business services.

Syntax

```
startBusinessService(processName,
                     moduleName,
                     version,
                     description,
                     parent);
```

```
startBusinessService(businessServices,
                     parent);
```

Parameters

- *processName* - (String) The business service process name.
- *moduleName* - (String) The module name of the business service.
- *version* - (String) The version of the business service.
- *description* - (String) (optional) A description that is displayed on the form caption bar.

- *parent* - (Object) (optional) Identifies the General Interface component where the business service form is to be loaded. If omitted, it is loaded in a dialog. Note that this parameter is ignored if more that one business service is passed in the method call.
- *businessServices* - (Array<Objects>) These identify the business services to be started. Each Object in the array must have the following properties, identifying an existing business service (see above for definitions):
  - *processName* - (String)
  - *moduleName* - (String)
  - *version* - (String)
  - *description* - (String) (optional)

Returns

none

startBusinessServiceWithData

This method starts a business service that has formal parameters.

This method has two signatures: one to start a single business service, and one to start multiple business services.

Syntax

```
startBusinessServiceWithData(processName,
                             moduleName,
                             version,
                             dataFeed,
                             description,
                             parent);
```

```
startBusinessServiceWithData(businessServices,
                             dataFeed,
                             parent);
```

Parameters

- *processName* - (String) The business service process name.
- *moduleName* - (String) The module name of the business service.
- *version* - (String) The version of the business service.
- *dataFeed* - (String) Data used to start the Business Service, in JSON format. An example of a JSON-formatted data feed is:

  ```
  ‘{ "items":[{"$param":"myText", "mode":"INOUT", "type":"STRING",
  "$value":"abc"}, {"$param":"myInt", "mode":"INOUT", "type":"NUMBER",
  "$value":"123"}]}’
  ```
- *description* - (String) (optional) A description that is displayed on the form caption bar.
- *parent* - (Object) (optional) Identifies the General Interface component where the business service form is to be loaded. If omitted, it is loaded in a dialog. Note that this parameter is ignored if more that one business service is passed in the method call.
- *businessServices* - (Array<Objects>) These identify the business services to be started. Each Object in the array must have the following properties, identifying an existing business service (see above for definitions):
  - *processName* - (String)
  - *moduleName* - (String)

– *version* - (String)
– *dataFeed* - (String)
– *description* - (String) (optional)

Returns

none

Event View Methods

A Tools Interface method is available for working with event views in your custom WCC application.

showEvents

This method causes the **EventViewer** component to create a temporary event view, based on the parameters passed in the **showEvents** method, and adds the view to the event view list.

Syntax

```
showEvents(eventLinkId,
           substitutionDataXml,
           menuText,
           eventLinkXml);
```

Parameters

- *eventLinkId* (String) - The <links> element **messageId** attribute value to match. (For information about event links and the <links> element, see the "Customizing Events" topic in the *TIBCO Workspace Configuration and Customization Guide.)*
- *substitutionDataXml* (String) - The XML containing the attributes that are substituted into the <link> element that is used. The root element name is record: <record att1=”val1” att2=”val2”/>
- *menuText* (String) (Optional) - If there are multiple <link> elements, the **menuText** value is used to locate a specific one. If null, the first link is used.
- *eventLinkXml* (String) (Optional) - The XML that defines the event links. If this is null, the application-configured **eventLinks** XML is used.

Returns

none

Notes

For an example of displaying a temporary event view in an IFrame, see wccLoginManagedHub Sample Application.

Also see com.tibco.wcc.schema.showEvents for information about publishing a “Version 2" of the **showEvent** schema to create a temporary event view.

User Options Methods

Tools Interface method is available for working with user options in your custom WCC application.

workspaceOptions

This method allows the user to view/modify his/her user options by displaying the Options dialog.

For information about the user options available from the Options dialog, see the *TIBCO Workspace User’s Guide*.

Syntax

```
workspaceOptions(parent);
```

Parameters

- *parent* - (Object) (optional) Identifies the General Interface component where the **Options dialog** is to be loaded. If omitted, it is loaded in a dialog.

Returns

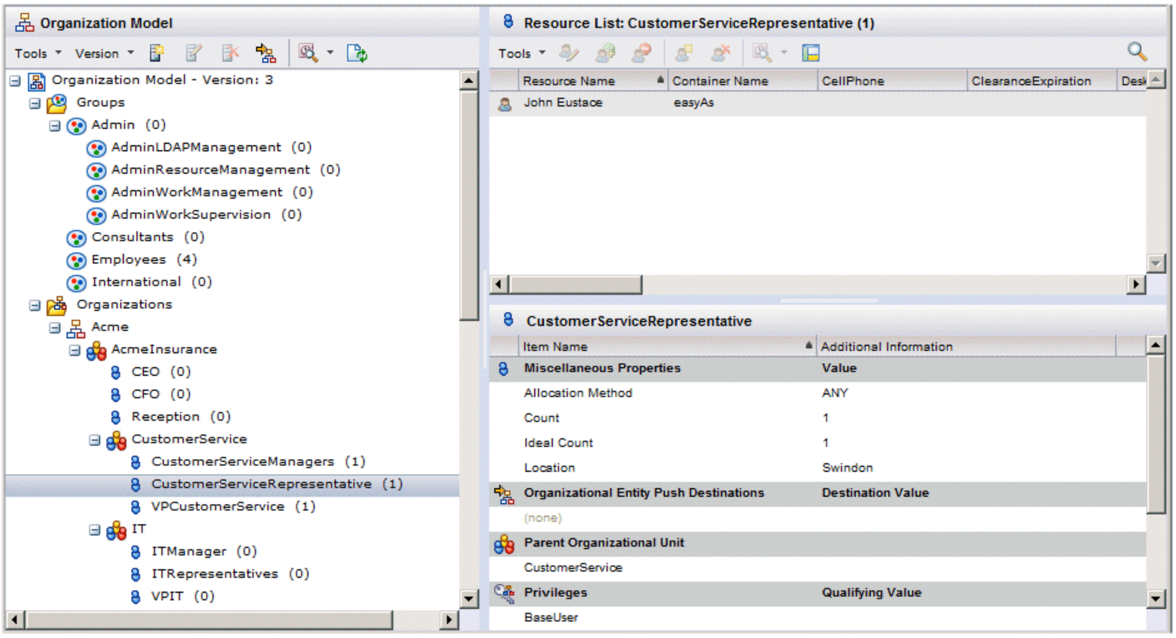
none

Organization Browser Methods

Tools Interface method is available for working with the Organization Browser in your custom WCC application.

workspaceOrganizationBrowserWindow

This method displays the Organization Browser, which can be used to create LDAP containers, as well as map resources to groups and/or positions in the Organization Model.


Organization Model
Tools
Version
Resource List: CustomerServiceRepresentative (1)
Tools
CustomerServiceRepresentative

```
Organization Model - Version: 3
  Groups
    Admin (0)
      AdminLDAPManagement (0)
      AdminResourceManagement (0)
      AdminWorkManagement (0)
      AdminWorkSupervision (0)
    Consultants (0)
    Employees (4)
    International (0)
  Organizations
    Acme
      AcmeInsurance
        CEO (0)
        CFO (0)
        Reception (0)
        CustomerService
          CustomerServiceManagers (1)
          CustomerServiceRepresentative (1)
          VPCustomerService (1)
        IT
          ITManager (0)
          ITRepresentatives (0)
          VPIT (0)
```

Resource Name	Container Name	CellPhone	ClearanceExpiration	Desk
John Eustace	easyAs			

Item Name	Additional Information
Miscellaneous Properties	**Value**
Allocation Method	ANY
Count	1
Ideal Count	1
Location	Swindon
Organizational Entity Push Destinations	**Destination Value**
(none)	
Parent Organizational Unit	
CustomerService	
Privileges	**Qualifying Value**
BaseUser	

For information about using the Organization Browser, see the *TIBCO Organization Browser User's Guide*.

Syntax

```
workspaceOrganizationBrowserWindow(parent);
```

Parameters

- *parent* - (Object) (optional) Identifies the General Interface component where the **Organization Browser** is to be loaded. If omitted, it is loaded in a dialog.

Returns

none

Configuration Methods

Tools Interface methods are available for configuring your custom WCC application.

configAdmin

This method displays the Configuration Administrator, which is used to configure things such as user access sets, in a deployed application.

Syntax

```
configAdmin(parent);
```

Parameters

- *parent* - (Object) (optional) Identifies the General Interface component where the Configuration Administrator is to be loaded. If omitted, it is loaded in a dialog.

Returns

none

Notes

A parameter exception results if the *parent* is defined, but does not implement the General Interface **setChild** method.

Locale Methods

Tools Interface methods are available for getting or setting the locale.

getAppLocales

This method returns an array of **jsx3.util.Locale** objects, one for each locale supported by the application and defined in the **locales** attribute of `JSXAPPS/base/locale/locale.xml`.

Syntax

```
getAppLocales();
```

Parameters

none

Returns

- Array of jsx3.util.Locale objects

setAppLocale

This method sets the locale of the application server. A "localeChanged" PageBus event is published to notify application components of the locale change so that they can reload and repaint as necessary to reflect the new language.

Syntax

```
setAppLocale(localeKey,
             restart);
```

Parameters

- *localeKey* - (String) The lowercase, two-letter ISO-639 language code (ll), and optionally the uppercase (CC), two-letter ISO-3166 country code if the locale is country specific. If the country code is included, it must be separated from the language code with an underscore, in the form "ll_CC" (e.g., "es_MX" for Mexican Spanish).

  For a list of language codes, visit the following web site:

  – `http://www.loc.gov/standards/iso639-2/langhome.html`

    For a list of country codes, visit the following web site:

  – `http://www.iso.org/iso/country_codes/iso_3166_code_lists/country_names_and_code_elements.htm`

- *restart* (Boolean) (Optional) Specifies whether or not the application needs to be restarted to apply the new locale.

  – If true, the new locale is not applied until the application is restarted. A message is displayed indicating that the application must be restarted.

  – If false, the new locale is applied immediately and application components are repainted to reflect the new language.

    Default=false

Returns

none

Mixin Interface Methods

The `AppMain.js` mixin interface is implemented by the Application class (`Application.js`). It contains samples of how custom methods can be added.

The Application class mixin interface is located as follows:

`*StudioHome*\wcc\*version*\JSXAPPS\*WCCProjectName*\application\js\AppMain.js`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.
- *WCCProjectName* is the name of the General Interface Builder project that contains your custom application. If you are working with the Workspace application, this is "workspace".

Custom code can be added to `AppMain.js` as needed.

The following example methods are provided in the Application class mixin interface.

customPostLoginExample

This method, which provides an example of performing a custom action post login, is called by the **postLogin** method in the Application class.

If you are creating your own custom WCC application, either delete or comment out the call to **customPostLoginExample** by the **postLogin** method in the Application class.

Also see: postLogin

customPostLogoutExample

This method, which provides an example of performing a custom action post logout, is called by the **postLogout** method in the Application class.

If you are creating your own custom WCC application, either delete or comment out the call to **customPostLogoutExample** by the **postLogout** method in the Application class.

Also see: postLogout

sampleToolbarHandler

This example method is used by the provided custom menus sample application. It is used to handle custom application code when the user makes a selection from a custom toolbar button.

For information about using this custom handler in the sample application, see the *Adding Custom Menus and Toolbar Buttons* in the *TIBCO Workspace Configuration and Customization* guide.

sampleMenuHandler

This example method is used by the provided custom menus sample application. It is used to handle custom application code when the user makes a selection from a custom menu.

For information about using this custom handler in the sample application, see the *Adding Custom Menus and Toolbar Buttons* in the *TIBCO Workspace Configuration and Customization* guide.

Component Public Methods

A number of the WCC components contain public methods that allow you to perform functions specific to that component.

These methods must be called from their respective component instance (e.g., com.tibco.wcc.components.DataMask).

DataMask Component Public Methods

Public methods are available for working with the DataMask component.

Also see: Data Mask Methods.

addCursorBlockId

This method adds the given *jsxObjId* to an array used to set the CSS cursor on objects included in this array.

When the DataMask is shown, the cursor is set to the value in this.showMaskCursorValue (default = “wait”).

When the DataMask is not shown, the cursor is set to the value in this.hideMaskCursorValue (default = “default”).

Syntax

```
addCursorBlockId(jsxObjId);
```

Parameters

jsxObjId - (String) The JSX-generated ID for the object.

Returns

none

setShowMaskCursorValue

This method sets the value used for the CSS cursor property when the DataMask is displayed.

Syntax

```
setShowMaskCursorValue(cursorValue);
```

Parameters

cursorValue - (String) The CSS value to use in setting the cursor.

Returns

none

setHideMaskCursorValue

This method sets the value used for the CSS cursor property when the DataMask is not displayed.

Syntax

```
setHideMaskCursorValue(cursorValue);
```

Parameters

cursorValue - (String) The CSS value to use in setting the cursor.

Returns

none

WorkItemsPreview Component Public Methods

Public methods are available for working with the WorkItemPreview component.

getCaption

This method returns the text that appears in the caption bar of the WorkItemsPreview component.

Syntax

```
getCaption();
```

Parameters

none

Returns

- String - Text in the caption bar.

showCaption

This method either shows or hides the caption bar on the WorkItemsPreview component.

Syntax

```
showCaption(bShow);
```

Parameters

bShow - (boolean) True = caption bar is displayed. False = caption bar is not displayed.

Returns

none

ProcessInstancesPreview Component Public Methods

Public methods are available for working with the ProcessInstancesPreview component.

getCaption

This method returns the text that appears in the caption bar of the ProcessInstancesPreview component.

Syntax

```
getCaption();
```

Parameters

none

Returns

- String - Text in the caption bar.

showCaption

This method either shows or hides the caption bar on the ProcessInstancesPreview component.

Syntax

```
showCaption(bShow);
```

Parameters

bShow - (boolean) True = caption bar is displayed. False = caption bar is not displayed.

Returns

none

EventViewer Component Public Methods

Public methods are available for working with the EventViewer component.

getCaption

This method returns the text that appears in the caption bar of the EventViewer component.

Syntax

```
getCaption();
```

Parameters

none

Returns

- String - Text in the caption bar.

showCaption

This method either shows or hides the caption bar on the EventViewer component.

Syntax

```
showCaption(bShow);
```

Parameters

bShow - (boolean) True = caption bar is displayed. False = caption bar is not displayed.

Returns

none

OrganizationResourceList Component Public Methods

Public methods are available for working with the OrganizationResourceList component.

getCaption

This method returns the text that appears in the caption bar of the OrganizationResourceList component.

Syntax

```
getCaption();
```

Parameters

none

Returns

- String - Text in the caption bar.

showCaption

This method either shows or hides the caption bar on the OrganizationResourceList component.

Syntax

```
showCaption(bShow);
```

Parameters

bShow - (boolean) True = caption bar is displayed. False = caption bar is not displayed.

Returns

none

BusinessServices Component Public Methods

Public methods are available for working with the BusinessServices component.

setFormContainerBlock

This method sets the formContainerBlock, which is used as the parent container for forms opened from this BusinesServices component if the application layout mode is not set to float, and **Dock Forms** is selected on the **View** menu.

This value must be set before the component is rendered — the call is ignored after the component is rendered.

Syntax

```
setFormContainerBlock(value);
```

Parameters

value - (jsx3.gui.Block) The parent block into which the docked form will be rendered.

Returns

None

getFormContainerBlock

This method returns the formContainerBlock, which is used as the parent container for forms opened from this BusinesServices component if the application layout mode is not set to float, and **Dock Forms** is selected on the **View** menu.

Syntax

```
getFormContainerBlock();
```

Parameters

None

Returns

- jsx3.gui.Block - The parent block into which the docked form will be rendered.

Prototype Public Methods

There is single public method in the `com.tibco.wcc.components.Prototype` class: getVersion.

getVersion

This method returns the version number of this instance.

Syntax

```
getVersion();
```

Parameters

none

Returns

- String

Server Requests

This topic describes making direct server requests from a custom WCC application. These requests provide a means of accessing certain types of BPM data that are normally only exposed through visual WCC components.

To demonstrate the use of server requests, a sample application, **wccApiSample**, is provided in the `\samples` directory when Workspace is installed with Business Studio. This sample application provides a UI that allows you to execute each of the available server requests, then receive a response to the request. For more information about setting up and using the **wccApiSample** application, see wccApiSample Application.

Submitting Server Requests

Requests are submitted to the server using the **executeServerRequests** WCC public method.

```
this.executeServerRequests(socketId,
                           arrRequests,
                           asyncClass,
                           asyncFunction);
```

where:

- *socketId* - (String) Information identifying this particular set of requests. The socketIds can be reused, although if the application will be making requests asynchronously, it must make sure no two outstanding requests have the same socketId.
- *arrRequests* (Array[Objects]) An array of request objects created using the server requests that are documented in Server Request Reference.

  More about creating the request object is provided below.
- *asyncClass* (Object) (optional) This is specified if the response is to be returned asynchronously. It identifies the class/object that contains the function that is called once the response has come back from the server.
- *asyncFunction* (String) (optional) This is specified if the response is to be returned asynchronously. It identifies the name of the function that is called once the response has come back from the server.

To execute a server request, you must first create a request object using one of the server requests (for a complete list of the available server requests, see Server Request Reference). For example:

```
var request = com.tibco.wcc.base.Requests.getUserPrivileges(requestId,
txtResourceGuid);
```

The parameters specified for each request vary, depending on the request — they provide the input for the server request. For instance, the *txtResourceGuid* parameter for the **getUserPrivileges** request shown above identifies the specific user whose privileges are to be returned.

Most parameters are simple strings, numbers, or boolean values, although in some cases, a parameter may be a JavaScript object with certain properties and values, or an array of strings or JavaScript objects.

All requests have *requestId* as one of the parameters. Multiple requests can be submitted to the server at one time, and it is also possible to make requests asynchronously. This requestId is echoed in the response to help your code match up the response with the request.

If you do not supply the optional parameters of *asyncClass* and *asyncFunction* in the call to **executeServerRequests**, the method call returns a **jsx3.xml.Document** object containing the server response.

If the optional parameters *are* specified, the method call returns nothing. Later, when the response arrives from the server, the specified function (*asyncFunction* parameter) in the specified class (*asyncClass* parameter) is called. The function should be set up with two parameters:

- *socketId* (String) The socketId that was submitted when **executeServerRequests** was called.
- *oResponseXml* (jsx3.xml.Document) A TIBCO General Interface XML Document that contains response information for all of the requests, or possible error information if any of the requests could not be successfully executed.

Submitting Request Examples

Examples are provided of setting up requests and submitting them to the server.

Synchronous Examples

In the following examples, the optional parameters are omitted, so they will wait for the server to respond, and they will return the response XML:

Example 1:

```
...
var request = com.tibco.wcc.base.Requests.ping(requestId);
var oResponseXml = this.executeServerRequests(socketId, [request]);
...
```

Example 2:

```
...
var oResponseXml = this.executeServerRequests(socketId,
[ com.tibco.wcc.base.Requests.ping(requestId) ] );
...
```

Example 3:

```
...
var requests = new Array();
requests.push(com.tibco.wcc.base.Requests.ping(requestId));
requests.push(com.tibco.wcc.base.Requests.
getUserPrivileges(requestId, txtResourceGuid));
var oResponseXml = this.executeServerRequests(socketId, requests);
...
```

Example 4:

```
...
var oResponseXml = this.executeServerRequests(socketId,
[ com.tibco.wcc.base.Requests.ping(requestId),
com.tibco.wcc.base.Requests.getUserPrivileges(requestId,
txtResourceGuid) ] );
...
```

Asynchronous Example

In the following example, the optional parameters are provided, causing it to be executed asynchronously.

```
...
classProto.submitServerRequest = function () {
   var requests = new Array();
   requests.push(com.tibco.wcc.base.Requests.ping(requestId));
   requests.push(com.tibco.wcc.base.Requests
   .getUserPrivileges(requestId, txtResourceGuid));
   this.executeServerRequests(socketId, requests, this,
   'handleServerResponse');
};
classProto.handleServerResponse = function (socketId, oResponseXml) {
   var txtReturnCode = oResponseXml.selectSingleNode
   ('/ap:ActionResult/ap:Status/ap:ReturnCode').getValue();
   var txtReturnComment = oResponseXml.selectSingleNode
   ('/ap:ActionResult/ap:Status/ap:ReturnComment').getValue();
   If (txtReturnCode != "0") {
   alert(txtReturnCode  + " - " + txtReturnComment );
   }
}
...
```

Since the function handling the response from the server is defined as part of the same class as the function executing the requests, the keyword 'this' can be used to identify the class.

The response XML is returned in a **jsx3.xml.Document** object so it can be examined or manipulated using the features of that object as described in the TIBCO General Interface documentation. In the code above, the return code and return comment are retrieved.

Responses to Server Requests

Once the server has processed all of the individual requests in a server request, it consolidates the data under a single XML root element, <**apActionResult**>. This XML response is then sent back to the application via HTTP.

The format of the XML response is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<ap:ActionResult xmlns:ap="http://tibco.com/bpm/actionprocessor">
   <ap:Status>
      <ap:Version></ap:Version>
      <ap:ReturnCode></ap:ReturnCode>
      <ap:ReturnComment></ap:ReturnComment>
      <ap:ReturnDateTime></ap:ReturnDateTime>
   </ap:Status>
   <ap:Requests>
      .
      .
      .
   </ap:Requests>
</ap:ActionResult>
```

Request-specific data (for example, information about the business service categories for a **ListBusinessServiceCategories** request) is included in the response under the <**ap:Requests**> element. For example:

```
<ap:Requests>
  <ap:ListBusinessServiceCategories Id="ApiSample.listBusinessServiceCategories">
    <ap:Categories>
       <ap:Category>
          <ap:Name>Acme</ap:Name>
          <ap:ChildCategories />
       </ap:Category>
       <ap:Category>
          <ap:Name>uncategorized</ap:Name>
          <ap:ChildCategories />
       </ap:Category>
       <ap:Category>
          <ap:Name>Sales</ap:Name>
          <ap:ChildCategories />
       </ap:Category>
       <ap:Category>
              .
              .
              .
    </ap:Categories>
  </ap:ListBusinessServiceCategories>
</ap:Requests>
```

Response Status

The XML response to a server request contains an <ap:Status> element that indicates whether or not the action was successfully processed. It also contains information to help determine the problem if an error does occur.

The <ap:Status> element provides the version number of the Action Processor and the date and time the response was returned to the application. For example:

```
      .
      .
      .
<ap:Status>
   <ap:Version>1.1.0</ap:Version>
   <ap:ReturnCode>0</ap:ReturnCode>
   <ap:ReturnComment><![CDATA[ The Action was processed successfully. Check
the individual Request Results for their status.</ap:ReturnComment>
   <ap:ReturnDateTime>2011-02-28T10:37:11.671-0800</ap:ReturnDateTime>
</ap:Status>
      .
      .
      .
```

A return code of 0 (zero) in the <ap:ReturnCode> element indicates there were no unexpected problems executing the requests.

Most other return codes are text, and are the same codes returned by Web Service calls. Refer to documentation for related Web Service calls for more details on those codes. An example is shown below:

```
      .
      .
      .
<ap:Status>
   <ap:Version>1.1.0 v19</ap:Version>
   <ap:ReturnCode>INVALID_ENTITY_REFERENCE_EXCEPTION</ap:ReturnCode>
   <ap:ReturnComment>Invalid organisational entity reference: [version: null, type:
RESOURCE, guid: _bnFngALvEeCHF4vrCq9WPw]</ap:ReturnComment>
   <ap:ReturnDateTime>2011-03-10T15:01:15.218-0800</ap:ReturnDateTime>
</ap:Status>
      .
      .
      .
```

The return comment will usually provide more detailed information about any problem encountered.

If multiple requests are submitted in one batch, execution of those requests on the server are handled one at a time, and if one request fails, the exception is returned and the remaining requests are not executed.

The value in the <ap:ReturnDateTime> element is in the form:

```
yyyy-mm-ddThh:nn:ss.uuu+/-oooo
```

where:

- yyyy-mm-dd = year, month, and day the XML response was sent
- T = separator between the date and time
- hh:nn:ss.uuu = hour, minute, second, and microsecond the XML response was sent
- +/- = indicates whether the offset from GMT is plus (+) or minus (-)
- oooo = offset (in 2400-hour format) from Greenwich Mean Time (GMT)

For example:

```
2006-04-26T08:02:02.146-0700
```

In this example, the “-0700” indicates minus 7 hours from GMT.

wccApiSample Application

A sample application, **wccApiSample**, is provided with Workspace that illustrates the usage of all of the available server requests.

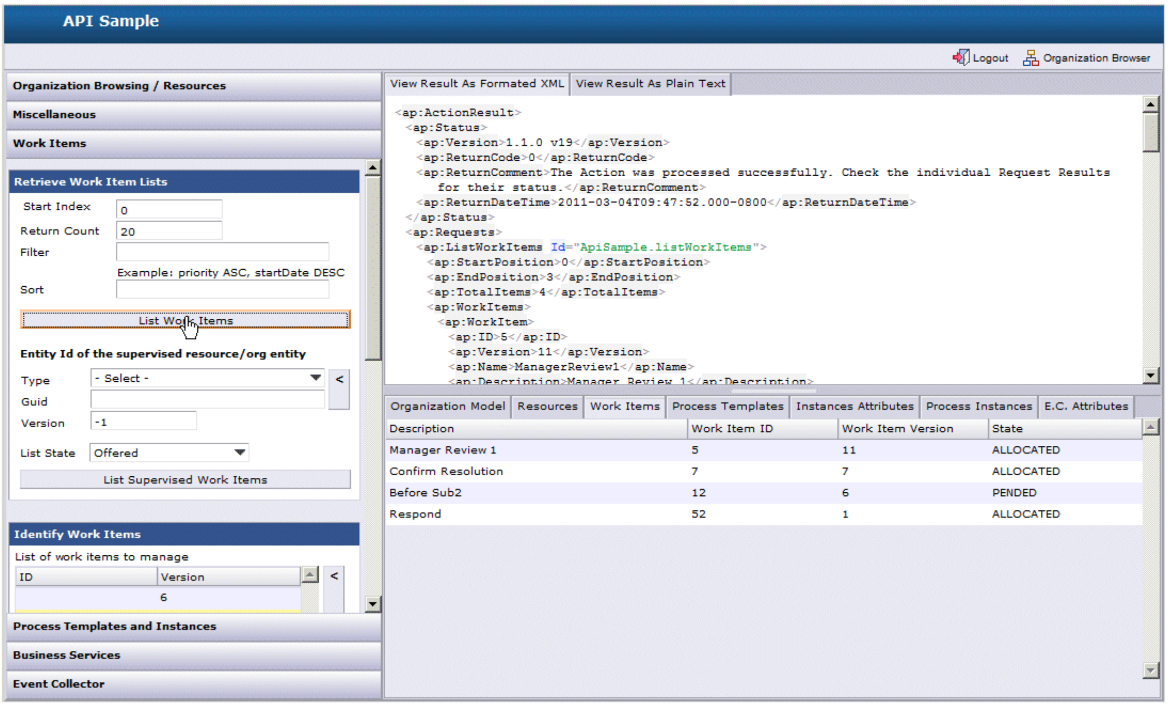
This sample application is provided in the following directory:

StudioHome\wcc\version\samples\wccApiSample\JSXAPPS\wccApiSample\

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.

This sample application provides a UI that allows you to execute a server request, then receive a response to the request:



Start Index	0
Return Count	20
Filter	
	Example: priority ASC, startDate DESC
Sort	

Entity Id of the supervised resource/org entity

Type	- Select -
Guid	
Version	-1
List State	Offered

List of work items to manage

ID	Version
	6

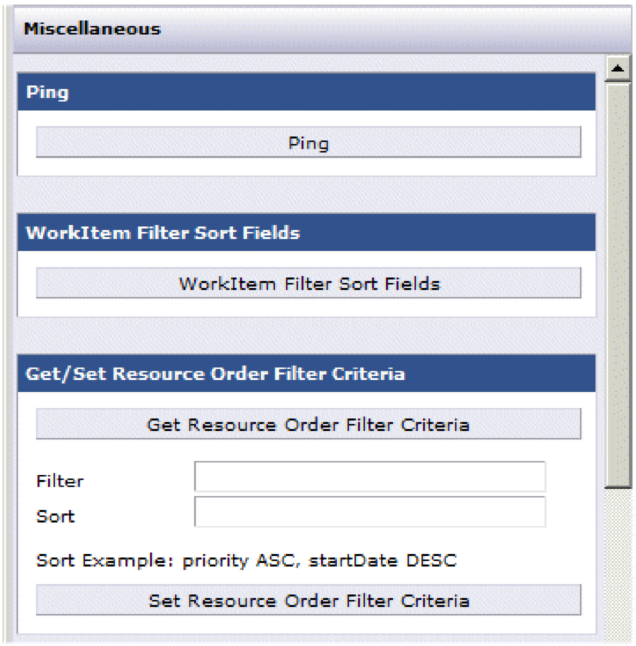
```
<ap:ActionResult>
 <ap:Status>
  <ap:Version>1.1.0 v19</ap:Version>
  <ap:ReturnCode>0</ap:ReturnCode>
  <ap:ReturnComment>The Action was processed successfully. Check the individual Request Results
    for their status.</ap:ReturnComment>
  <ap:ReturnDateTime>2011-03-04T09:47:52.000-0800</ap:ReturnDateTime>
 </ap:Status>
 <ap:Requests>
  <ap:ListWorkItems Id="ApiSample.listWorkItems">
   <ap:StartPosition>0</ap:StartPosition>
   <ap:EndPosition>3</ap:EndPosition>
   <ap:TotalItems>4</ap:TotalItems>
   <ap:WorkItems>
    <ap:WorkItem>
     <ap:ID>5</ap:ID>
     <ap:Version>11</ap:Version>
     <ap:Name>ManagerReview1</ap:Name>
     <ap:Description>Manager Review 1</ap:Description>
```

Description	Work Item ID	Work Item Version	State
Manager Review 1	5	11	ALLOCATED
Confirm Resolution	7	7	ALLOCATED
Before Sub2	12	6	PENDED
Respond	52	1	ALLOCATED

When run, the **wccApiSample** application displays a Login dialog, requiring a valid user name and password to be able to use the sample application.

The wccApiSample dialog consists of three panes:

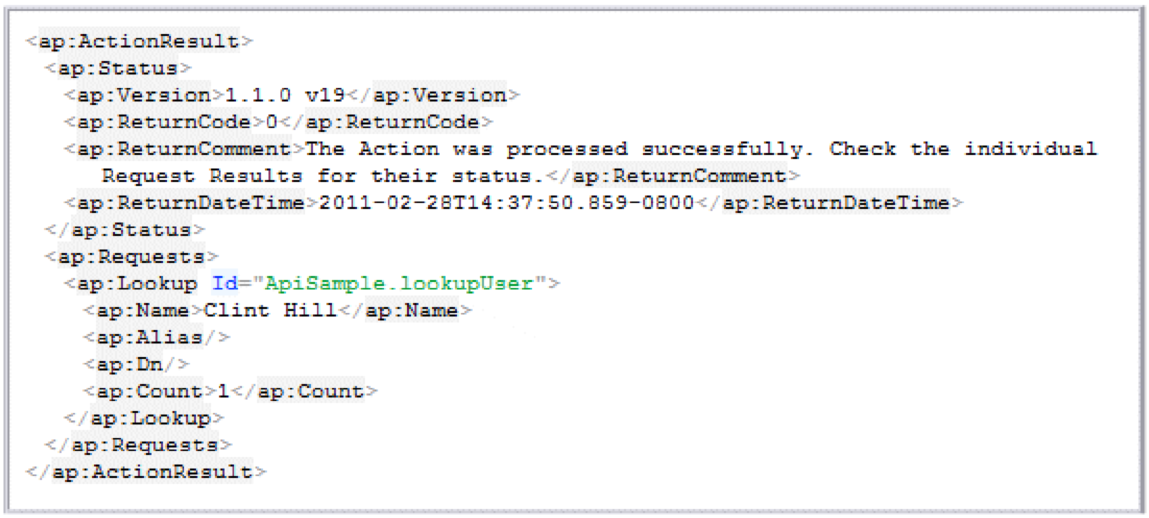
- The left pane consists of a series of buttons (**Organization Browsing / Resources**, **Miscellaneous**, **Work Items**, etc.), that when clicked, display additional buttons and input fields. For example, the following is displayed when the **Miscellaneous** button is clicked:


Miscellaneous
Ping
Ping
WorkItem Filter Sort Fields
WorkItem Filter Sort Fields
Get/Set Resource Order Filter Criteria
Get Resource Order Filter Criteria
Filter
Sort
Sort Example: priority ASC, startDate DESC
Set Resource Order Filter Criteria

Each of these buttons (for example, **Ping**) causes a request to be sent to the server. The input fields are used to enter the parameters required by the request.

For information about each of the available requests, and their input parameters, see Server Request Reference.

- The upper-right pane contains the response from the server for the request that was submitted as a result of clicking a button in the left pane. For example:

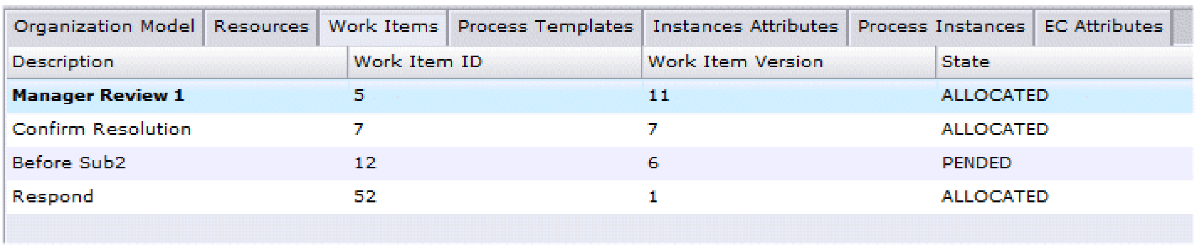


```
<ap:ActionResult>
  <ap:Status>
    <ap:Version>1.1.0 v19</ap:Version>
    <ap:ReturnCode>0</ap:ReturnCode>
    <ap:ReturnComment>The Action was processed successfully. Check the individual
      Request Results for their status.</ap:ReturnComment>
    <ap:ReturnDateTime>2011-02-28T14:37:50.859-0800</ap:ReturnDateTime>
  </ap:Status>
  <ap:Requests>
    <ap:Lookup Id="ApiSample.lookupUser">
      <ap:Name>Clint Hill</ap:Name>
      <ap:Alias/>
      <ap:Dn/>
      <ap:Count>1</ap:Count>
    </ap:Lookup>
  </ap:Requests>
</ap:ActionResult>
```

This pane contains tabs that allow you to view the response in either formatted XML or plain text.

For information about server responses, see Responses to Server Requests.

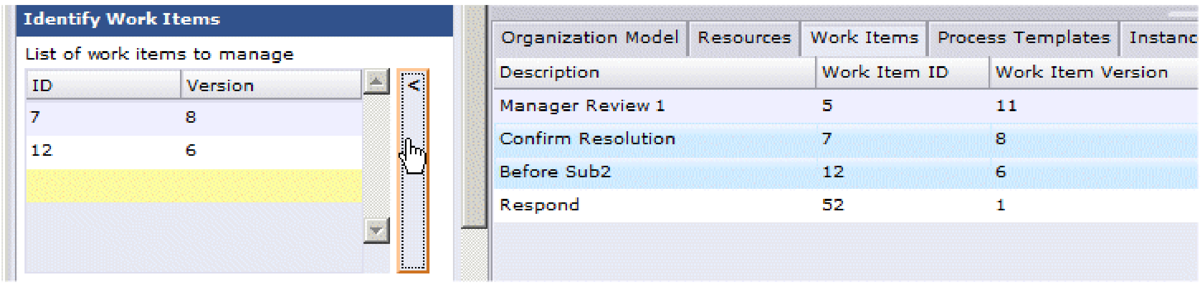
- The lower-right pane consists of a number of tabs that display information as a result of an action request. The appropriate tab is displayed when a request returns information. For example, if the **List Work Items** button is clicked (which executes the **listWorkItems** request), the **Work Items** tab in the lower-right pane lists the work items that are returned:



Organization Model	Resources	Work Items	Process Templates	Instances Attributes	Process Instances	EC Attributes

Description	Work Item ID	Work Item Version	State
Manager Review 1	5	11	ALLOCATED
Confirm Resolution	7	7	ALLOCATED
Before Sub2	12	6	PENDED
Respond	52	1	ALLOCATED

Note that some of the action requests allow you to make a selection in a list in the lower-right pane, then move information about the selected item(s) to input fields in the left pane using the provided < button:



Identify Work Items

List of work items to manage

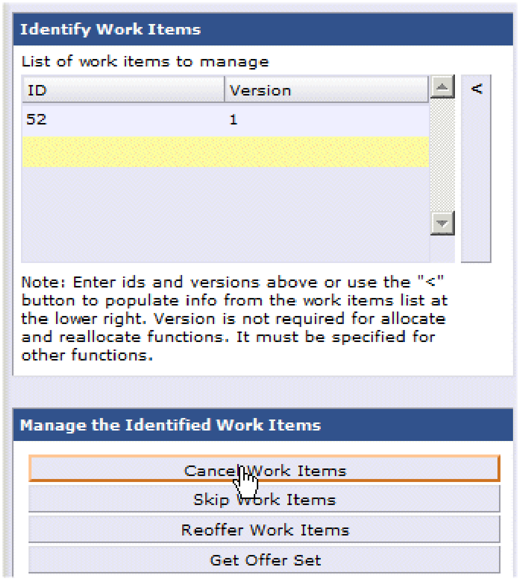
ID	Version
7	8
12	6

Organization Model	Resources	Work Items	Process Templates	Instanc

Description	Work Item ID	Work Item Version
Manager Review 1	5	11
Confirm Resolution	7	8
Before Sub2	12	6
Respond	52	1

Also, some of the action requests in the **wccApiSample** application require that you identify the item(s) (for example, work item(s) or process instance(s)) by moving them into a list like the **Identify Work Items** list shown in the example above. Once the list is populated with the desired item(s), you can initiate the desired request.

For example, to cancel a work item, populate the work item list in the lower-right pane (by executing the **List Work Items** or **List Supervised Work Items** request), select the work item from the list, populate the **Identify Work Items** list (by clicking the < button, then click the **Cancel Work Items** button to execute the request:



Identify Work Items

List of work items to manage

ID	Version
52	1

Note: Enter ids and versions above or use the "<" button to populate info from the work items list at the lower right. Version is not required for allocate and reallocate functions. It must be specified for other functions.

Manage the Identified Work Items

Setting Up the wccApiSample Application

Procedure

1. Copy the following directory...

   `StudioHome\wcc\version\samples\wccApiSample\JSXAPPS\wccApiSample`

   ... to the following directory:

   `StudioHome\wcc\version\JSXAPPS\`

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.
2. Open the wccApiSample application's configuration file, which is located as follows:

   `StudioHome\wcc\version\JSXAPPS\wccApiSample\config.xml`
3. Locate the **ActionProcessors** record in the `config.xml` file.
4. Set the **baseUrl** attribute to the URL of the Action Processor. The string in the **baseUrl** attribute must be in the form:

   `http://Host:Port/bpm/actionprocessor/actionprocessor.servlet`

   where:

   - *Host* is the name of the machine hosting the BPM runtime
   - *Port* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

     Note that the **weighting** attribute is not used at this time.

     You might also want to also set the **disable** attribute in the **SessionMonitor** record to "true", which will prevent the application from timing out while you are using it.
5. Save and close the `config.xml` file.
6. Copy the wccApiSample application launch fragment from here...

   `StudioHome\wcc\version\Samples\wccApiSample\wccApiSample.html`

   ... to here:

   `StudioHome\wcc\version\`

Running the wccApiSample Application

Procedure

1. Execute the wccApiSample launch fragment:

   `StudioHome\wcc\version\wccApiSample.html`

   The Login dialog is displayed.
2. Enter a valid user name and password, then click **OK**.

   The ApiSample dialog is displayed.

   For information about using the buttons and fields on the ApiSample dialog to send requests to the Action Processor, see wccApiSample Application.

Deploying the wccApiSample Application

A `wccApiSample.create.war.cmd` file is provided that can be used to create a WAR file that can be used to deploy the sample application to a runtime node.

For more information, see the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Configuration and Customization* guide.

Application Class for wccApiSample

The **Application** class (**com.tibco.wcc.wccApiSample.Application**) provided with the wccApiSample application illustrates calling each of the server requests, as well as submitting the request, and handling the response from the server.

The **Application** class contains four sections specifically related to submitting server requests. They are labelled Section 1, 2, 3, and 4 in the file (`wccApiSample\application\js\Application.js`), and are described below:

Section 1

This section contains the functions that are called when a server request is initiated by clicking a button in the left pane of the wccApiSample application.

As an example, the following function is called when the **Allocate Work Items** button is clicked in the wccApiSample application:

```
classProto.execute_allocateWorkItems = function () {
   var requestId = "ApiSample.allocateWorkItems";
   var socketId = requestId;
   var arrayWorkItems = this.private_getWorkItems();
   var pane =
this.getAppBlock().getDescendantOfName('blockEntity_allocateWorkItems');
   var txtResourceGuid = pane.getDescendantOfName('txtGuid').getValue();
   var request =
com.tibco.wcc.base.Requests.allocateWorkItems(requestId,                   arrayWork
Items, txtResourceGuid);
   var oResponseXml = this.executeServerRequests(socketId, [request]);
   this.displayResponseXml(oResponseXml);
```

Most of the functions in Section 1 illustrate building the request object, then submitting the request with the **executeServerRequests** method.

Section 2

This section contains helper functions that the sample application uses to process the server request responses. Responses are in an XML format that is displayed in its entirety in the upper-right portion of the sample application. For some requests, the response XML is used to populate a more readable list of data in a General Interface Matrix control.

Section 3

This section contains helper functions that the sample application uses to populate complex input for some of the server requests. Valid input for some requests must come from the response of another request. The sample application displays some results in a list where items can be selected. In places where another server request can use that information as input, there is a button that moves the appropriate data about the selected items into the input. For instance, once you have a list of process templates, you can select one or more templates and add them to the input for retrieving process instances. The functions in this section perform that move.

Section 4

This section contains helper functions that the sample application uses to generate some of the more complex forms of input for the Server Requests. Some of the inputs are arrays of items. These functions build those arrays from user input controls.

You can use the functions in Sections 2-4 as a guide for handling user input data and request responses.

Server Request Reference

This topic describes all of the server requests available in Workspace.

The requests are organized in the same categories as they are in the left pane of the wccApiSample application (see wccApiSample Application).

Note that the server requests (including input parameters and the contents of the server response) are subject to change in future releases.

Also note that some of the items in the lists below are actually base Application-class methods. They are shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods. For information about the base Application-class methods, see Application Class Methods .

The following is a summary of the available requests. Click the request name to view details about the request.

- Organization Browsing / Resource Requests
  - **listModelVersions** - Returns a list of the organization model versions currently deployed.
  - **listOrgModelOverview** - Returns a set of organizational entities, nested as they exist in the organizational model.
  - **listMappedEntities** - Returns a list of resources that are explicitly assigned to the specified group or position.
  - **getResourceDetail** - Returns detailed information about a resource.
  - **lookupUser** - Determines whether or not a resource exists. If the resource exists, details of the resource can be returned.
  - **listQueryResources** - Returns a list of resources matching a Resource Query Language (RQL) statement.
- Miscellaneous Requests
  - **Ping** - This request verifies the server is responsive.
  - **workItemFilterSortFields** - Returns a list of fields that can be used to sort or filter a work item list.
  - **getResourceOrderFilterCriteria** - Returns the default sort and filter criteria stored on the server for the currently logged on user.
  - **setResourceOrderFilterCriteria** - Sets the default sort and filter criteria stored on the server for the currently logged on user.
  - **getUserPrivileges** - Returns a list of the privileges held by the specified user.
  - **listSystemActionAuthorizedEntities** - Returns a list of GUIDs identifying the organizational entities that are granted the specified system action.
  - **isActionAuthorized** - Returns true or false, indicating whether the specified system action is authorized.
- Work Item Requests

  - **listWorkItems** - Returns a list of the work items for the logged-in user.
  - **listSupervisedWorkItems** - Returns a list of work items for another resource or for an organizational entity (group, position, organizational unit, or organization).
  - **cancelWorkItems** - Returns currently opened work items to an unopened state. (This is actually an Application-class method — see the note above.)
  - **skipWorkItems** - Marks the specified work item as complete, removes it from the work item list, and causes the process to advance. It has the same affect as opening the work item and submitting it. (This is actually an Application-class method — see the note above.)
  - **reofferWorkItems** - Re-offers the specified work item. (This is actually an Application-class method — see the note above.)
  - **getOfferSet** - Returns a list of resources offered a set of work items.
  - **allocateWorkItems** - Allocates the specified work items to the specified resource.
  - **reallocateWorkItems** - Takes allocated work items and changes the allocation to another resource, optionally cancelling changes made by the original resource.
  - **pendWorkItems** - Hides allocated work items until a specified date / time is reached or until a specified period of time has elapsed. (This is actually an Application-class method — see the note above.)

- Process Template and Instance Requests

  - **listStartTemplates** - Returns a list of process templates that require no input parameters, so they can be started directly without having to invoke a business service that might include forms.
  - **startProcessInstance** - Starts an instance of a process template that requires no input parameters. (This is actually an Application-class method — see the note above.)
  - **queryProcessTemplateCount** - Returns the number of process templates that exist.
  - **listProcessTemplateAttributes** - Returns a list of attributes that can be used for filtering or sorting process templates.
  - **queryProcessTemplates** - Returns a list of process templates.
  - **listProcessInstanceAttributes** - Returns the attributes associated with instances of a particular process template.
  - **queryProcessInstanceCount** - Returns the number of instances that exist for the specified process template(s).
  - **queryProcessInstances** - Returns the instances for the specified process templates.
  - **queryProcessInstancesNextPage** - Returns the next page of process instances — used after calling the **queryProcessInstances** request with the page size set to a value other than 0.
  - **queryProcessInstancesPreviousPage** - Returns the previous page of process instances — used after calling the **queryProcessInstances** request with the page size set to a value other than 0, then calling the **queryProcessInstancesNext** request to get the next page of process instances.
  - **queryProcessInstancesFirstPage** - Returns the first page of process instances — used after calling the **queryProcessInstances** request with the page size set to a value other than 0.
  - **queryProcessInstancesLastPage** - Returns the last page of process instances — used after calling the **queryProcessInstances** request with the page size set to a value other than 0, then calling the **queryProcessInstancesNext** request to get the next page of process instances.
  - **queryDone** - Notifies the server that no more next/previous page requests will be made for a paged list of process instances. This frees up server-side resources for managing the list.
  - **suspendProcessInstance** - Suspends the specified process instance. (This is actually an Application-class method — see the note above.)

- **resumeProcessInstance** - Resumes (unsuspends) the specified suspended process instance. (This is actually an Application-class method — see the note above.)
- **cancelProcessInstance** - Cancels the specified process instance. (This is actually an Application-class method — see the note above.)
- **resumeHaltedProcessInstance** - Resumes a halted process instance so that the instance itself goes into a failed state because of the failed task. (This is actually an Application-class method — see the note above.)
- **retryProcessInstance** - Causes the previously failed task to be retried so that the halted process instance can progress normally. (This is actually an Application-class method — see the note above.)
- **ignoreProcessInstance** - Causes the failed task to be skipped in the halted process instance. The process instance continues processing from the point in the process after the failed task. (This is actually an Application-class method — see the note above.)

- Business Service Requests
  - **listBusinessServiceCategories** - Returns a list of all valid business service categories.
  - **queryBusinessServices** - Returns a list of business services for a specified category.
  - **queryBusinessServiceCategories** - Returns a list of business services for a specified category.
- Event Collector Requests
  - **eventCollectorGetComponents** - Returns a list of the component IDs used in event collector entries.
  - **eventCollectorGetAllAttributes** - Returns a list of all available event collector attributes.
  - **eventCollectorGetAttributes** - Returns a list of all event collector attributes associated with a specified component.
  - **eventCollectorQuery** - Returns a list of events that match a specified filter expression.

Organization Browsing Resource Requests

Server requests are available for working with organization models and resources.

listModelVersions

This request returns a list of the organization model versions currently deployed.

Syntax

```
com.tibco.wcc.base.Requests.listModelVersions(requestId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

An <ap:OrgModelVersion> element is returned for each organization model version that is currently deployed. The <ap:OrgModelVersion> element contains the following elements:

- <ap:Guid> - The GUID for this particular version of the organization model.
- <ap:Version> - The major version number of the organization model.
- <ap:Detail> - Contains details such as minor and micro version numbers, name, date deployed, etc.

For example:

```
<ap:Requests>
  <ap:VersionList Id="ApiSample.listModelVersions">
    <ap:OrgModelVersions>
      <ap:OrgModelVersion>
         <ap:Guid>0</ap:Guid>
         <ap:Version>0</ap:Version>
         <ap:Detail><![CDATA[<xml-fragment model-version="0"><deployment major="0"
             minor="1" micro="0" qualifier="" name="sys_org_model"
            deployed="2011-04-22T17:58:59.270Z"/></xml-fragment>
        </ap:Detail>
      </ap:OrgModelVersion>
                 .
                 .
                 .
```

listOrgModelOverview

This request returns a set of organizational entities, nested as they exist in the organizational model.

Syntax

```
com.tibco.wcc.base.Requests.listOrgModelOverview(requestId,
                                                 modelGuid);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *modelGuid* - (String) The GUID of the organization model version to be returned. Passing -1 causes the latest version to be returned.

Returns

An <ap:Node> element is returned for each entity in the organization model. These nodes are nested according to the hierarchy of the organization model (for instance, positions are subordinate to organization units). The <ap:Node> element contains the following elements:

- <ap:Type> - The type of organizational entity: “ORGANIZATION”, “ORGANIZATIONAL_UNIT”, “POSITION”, “GROUP”, or “LOCATION”.
- <ap:Guid> - The GUID for the organizational entity.
- <ap:Name> - The name of the organizational entity.
- <ap:AllocationMethod> - Indicates how work items are allocated to user. This is either one of the following:
  – ANY - Random
  – NEXT - Round Robin
- <ap:LocationGuid> - The GUID of the location to which the organizational entity is associated.
- <ap:ResourceCount> - The number of resources currently mapped to the organizational entity.
- <ap:IdealNumber> - The ideal number of resources that should be assigned to the position (only applicable to positions) as defined in TIBCO Business Studio.

For example:

```
<ap:Requests>
  <ap:ListOrgModelOverview Id="ApiSample.listOrgModelOverview">
    <ap:OrgModelOverview ModelVersion="-1">
      <ap:Node>
        <ap:Type>ORGANIZATION</ap:Type>
        <ap:Guid>_iPYXscpPEd64gM7QE8RwxA</ap:Guid>
        <ap:Name>Acme</ap:Name>
        <ap:AllocationMethod>ANY</ap:AllocationMethod>
        <ap:LocationGuid>_gqt1EMpREd64gM7QE8RwxA</ap:LocationGuid>
        <ap:ResourceCount>0</ap:ResourceCount>
        <ap:IdealNumber>0</ap:IdealNumber>
        <ap:Node>
          <ap:Type>ORGANIZATIONAL_UNIT</ap:Type>
          <ap:Guid>_oA-4AMpREd64gM7QE8RwxA</ap:Guid>
          <ap:Name>AcmeInsurance</ap:Name>
          <ap:AllocationMethod>ANY</ap:AllocationMethod>
          <ap:LocationGuid>_iqRSAMpREd64gM7QE8RwxA</ap:LocationGuid>
          <ap:ResourceCount>0</ap:ResourceCount>
          <ap:IdealNumber>0</ap:IdealNumber>
          <ap:Node>
            <ap:Type>ORGANIZATIONAL_UNIT</ap:Type>
            <ap:Guid>_sXK1IMpREd64gM7QE8RwxA</ap:Guid>
            <ap:Name>IT</ap:Name>
            <ap:AllocationMethod>ANY</ap:AllocationMethod>
            <ap:LocationGuid>_gqt1EMpREd64gM7QE8RwxA</ap:LocationGuid>
            <ap:ResourceCount>0</ap:ResourceCount>
            <ap:IdealNumber>0</ap:IdealNumber>
                 .
                 .
                 .
    </ap:OrgModelOverview>
  </ap:ListOrgModelOverview>
</ap:Requests>
```

listMappedEntities

This request returns a list of resources that are explicitly mapped to the specified group or position (as those are the only types of organizational entities to which resources can be mapped).

Syntax

```
com.tibco.wcc.base.Requests.listMappedEntities(requestId,
                                               entityType,
                                               guid,
                                               modelVersion,
                                               qualifier);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *entityType* - (String) GROUP or POSITION.
- *guid* - (String) The GUID of the group or position.
- *modelVersion* - (String) (optional) The version number of the organization model for the group or position. Passing -1 or no value causes it to use the latest version.
- *qualifier* - (String) (optional) Not applicable.

Returns

An <ap:ResourceDetail> element is returned for each resource that is mapped to the specified group or position.

For example:

```
<ap:Requests>
   <ap:ListMappedEntities Id="ApiSample.listMappedEntities">
      <ap:ResourceDetail>
            .
            .
            .
     </ap:ResourceDetail>
            .
```

```
             .
             .
```

For information about the elements contained in the <ap:ResourceDetail> element, see getResourceDetail.

getResourceDetail

This request returns detailed information about one or more resources.

Syntax

```
com.tibco.wcc.base.Requests.getResourceDetail(requestId,
                                              entities);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *entities* - (Array[Object]) Identifies the resource(s) whose detail information is to be returned. Note that the wccApiSample application requests the detailed information for a single resource, although the request can ask for detailed information from any number of resources by including more entities (resources) in the array:

  ```
  entities = [
    {
       type: entityType,
       guid: guid,
       modelVersion: modelVersion,
       qualifier: qualifier
    }
  ]
  ```

  where:

- *entityType* - (String) Describes the type of entity. This must be RESOURCE.
- *guid* - The GUID of the resource whose details are being requested.
- *modelVersion* - (String) (optional) The version number of the organization model for the group or position. Passing -1 or no value causes it to use the latest version.
- *qualifier* - (String) (optional) This is not applicable in this context, and is explicitly set to an empty string internally.

Returns

An <ap:ResourceDetail> element is returned for each resource that is mapped to the group or position. The <ap:ResourceDetail> element contains the following elements:

- <ap:Guid> - The GUID for the resource.
- <ap:Name> - The name of the resource.
- <ap:Type> - The resource type. In this context, it is HUMAN.
- <ap:ContainerId> - The ID of the LDAP container in which the resource was created.
- <ap:ContainerName> - The name of the LDAP container in which the resource was created.
- <ap:StartDate> - Not used at this time.
- <ap:EndDate> - Not used at this time.
- <ap:Alias> - The alias of the LDAP container in which the resource was created.
- <ap:Dn> - The distinguished name for the resource.

- <ap:Invalid> - true = the resource is considered invalid (the LDAP entry used to create the resource may have been deleted, or multiple resources were found in a secondary LDAP source that match the resource in the primary LDAP source. false = the resource is a valid resource.
- <ap:InvalidReason> - If <ap:Invalid> is true, this contains the reason the resource is invalid.
- <ap:GROUP> or <ap:POSITION> - One of these elements is included in the response for each group and position to which the resource is mapped. These elements contain the following elements:
  - <ap:Guid> - The GUID for the group or position.
  - <ap:EntityType> - The type of entity: GROUP or POSITION
  - <ap:ModelVersion> - The version of the organization model in which the group or position is defined.
  - <ap:Name> - The name of the group or position.
  - <ap:Qualifier> - This is not applicable in this context.
  - <ap:StartDate> - The date on which the resource's membership in the group or position takes effect.
  - <ap:EndDate> - The date one which the resource's membership in the group or position ends.
- <ap:PRIVILEGE> - One of these elements is included in the response for each privilege possessed by the resource. This element contains the following elements:
  - <ap:Guid> - The GUID for the privilege.
  - <ap:EntityType> - The type of entity: PRIVILEGE
  - <ap:ModelVersion> - The version of the organization model in which the privilege is defined.
  - <ap:Name> - The name of the privilege.
  - <ap:Qualifier> - A qualifier assigned to the privilege.
  - <ap:Origin> - This element contains the following sub-elements, which provide information about the group or position from which the resource obtained the privilege.
  - <ap:Guid> - The GUID of the group or position.
  - <ap:ModelVersion> - The version of the organization model in which the group or position is defined.
  - <ap:Name> - The name of the group or position.
  - <ap:Qualifier> - This is not applicable in this context.
  - <ap:StartDate> - The date on which membership in the group or position takes effect.
  - <ap:EndDate> - The date on which membership in the group or position is no longer in effect.
- <ap:TemporaryResourceDefinition> - A value is returned in this element, although it is not applicable when viewing resource information (it is only valid for *potential* resources, which are not returned by any of the server requests described here).
- <ap:HasDuplicates> - This is applicable only to potential resources, which are not returned by any of the public server requests, therefore, this value is not applicable in this context.

For example:

```
<ap:Requests>
   <ap:Resources Id="ApiSample.getResourceDetail">
      <ap:ResourceDetail>
         <ap:Guid>E3A3FC77-10FD-4326-9F56-FE230C6FC579</ap:Guid>
         <ap:Name>Liam Lawrence</ap:Name>
         <ap:Type>HUMAN</ap:Type>
         <ap:ContainerId>1</ap:ContainerId>
         <ap:ContainerName>easyAs</ap:ContainerName>
         <ap:StartDate />
         <ap:EndDate />
         <ap:Alias>easyAs</ap:Alias>
         <ap:Dn>OU=Liam Lawrence, OU=London, OU=AllEmployees, O=easyAsInsurance</
ap:Dn>
         <ap:Invalid>false</ap:Invalid>
         <ap:InvalidReason />
         <ap:GROUP>
              .        See separate <ap:GROUP> example below.
              .
         </ap:GROUP>
         <ap:POSITION>
              .        See separate <ap:POSITION> example below.
              .
         </ap:POSITION>
         <ap:PRIVILEGE>
              .        See separate <ap:PRIVILEGE> example below.
              .
         </ap:PRIVILEGE>
         <ap:TemporaryResourceDefinition>&lt;definition
container_id="1" primary_dn="OU=Liam Lawrence,
               OU=London, OU=AllEmployees,O=easyAsInsurance" /&gt;</
ap:TemporaryResourceDefinition>
         <ap:HasDuplicates>false</ap:HasDuplicates>
      </ap:ResourceDetail>
   </ap:Resources>
</ap:Requests>
```

<ap:GROUP> Example:

```
<ap:GROUP>
   <ap:Guid>_Jb8oYMpREd64gM7QE8RwxA</ap:Guid>
   <ap:EntityType>GROUP</ap:EntityType>
   <ap:ModelVersion>3</ap:ModelVersion>
   <ap:Name>Employees</ap:Name>
   <ap:Qualifier />
   <ap:StartDate />
   <ap:EndDate />
</ap:GROUP>
```

<ap:POSITION> Example:

```
<ap:POSITION>
   <ap:Guid>__0368MpREd64gM7QE8RwxA</ap:Guid>
   <ap:EntityType>POSITION</ap:EntityType>
   <ap:ModelVersion>3</ap:ModelVersion>
   <ap:Name>SalesManagers</ap:Name>
   <ap:Qualifier />
   <ap:StartDate />
   <ap:EndDate />
</ap:POSITION>
```

<ap:PRIVILEGE> Example:

```
<ap:PRIVILEGE>
   <ap:Guid>_P6ARUF-IEd-rno5lLiXABg</ap:Guid>
   <ap:EntityType>PRIVILEGE</ap:EntityType>
   <ap:ModelVersion>3</ap:ModelVersion>
   <ap:Name>WorkSupervisionSystemActions</ap:Name>
   <ap:Qualifier />
   <ap:Origin>
   <ap:Guid>_T1Fq0ALvEeCHF4vrCq9WPw</ap:Guid>
   <ap:ModelVersion>3</ap:ModelVersion>
   <ap:Name>AdminWorkSupervision</ap:Name>
   <ap:EntityType>GROUP</ap:EntityType>
   <ap:Qualifier />
   <ap:StartDate />
   <ap:EndDate />
   </ap:Origin>
</ap:PRIVILEGE>
```

lookupUser

This request determines whether or not a resource exists. If the resource exists, details of the resource can be returned.

Syntax

```
com.tibco.wcc.base.Requests.lookupUser(requestId,
                                       txtResourceName,
                                       chkReturnDetail);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *txtResourceName* - (String) The name of the resource (login name) being looked up.
- *chkReturnDetail* - (boolean) Indicates whether or not to return detail information about the resource.

Returns

Returns an <ap:Lookup> element, which contains the following elements:

- <ap:Guid> - The GUID for the resource.
- <ap:Name> - The name of the resource.
- <ap:Dn> - The distinguished name for the resource.
- <ap:Count> - The number of matching records found, as follows:
  - 0 if the name was not found.
  - 1 if the name was found.
- <ap:ResourceDetail> - This element, which contains sub-elements that provide details about the resource, is included in the response only if true is passed in *chkReturnDetail* in the request. For information about the contents of this element, see getResourceDetail.

For example:

```
<ap:Requests>
   <ap:Lookup Id="ApiSample.lookupUser">
      <ap:Name>Clint Hill</ap:Name>
      <ap:Alias />
      <ap:Dn />
      <ap:Count>1</ap:Count>
      <ap:ResourceDetail>
            .
            .
            .
```

listQueryResources

This request returns a list of resources matching a Resource Query Language (RQL) statement. The resource's name and GUID are always returned. You can also optionally request full resource details, as well as an organization model version from which the resources will be obtained.

Syntax

```
com.tibco.wcc.base.Requests.listQueryResources(requestId,
                                               txtQuery,
                                               chkReturnDetail,
                                               modelVersion);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *txtQuery* - (String) The RQL statement used to identify the resource. For information about RQL statements, see the *TIBCO Business Studio BPM Implementation* guide.
- *chkReturnDetail* - (boolean) Indicates whether or not to return detail information about the resource.
- *modelVersion* - (String) (optional) The version number of the organization model for the resources. Passing -1 or no value causes it to use the latest version.

Returns

The XML returned depends on whether or not detail information is requested (*chkReturnDetail*).

If *chkReturnDetail* = false, an <ap:Resource> element is returned, which contains the following elements:

- <ap:Name> - The name of the resource.
- <ap:Guid> - The GUID for the resource.

For example:

```
<ap:Requests>
  <ap:Resources Id="ApiSample.listQueryResources">
    <ap:Query>resource(name="Clint Hill")</ap:Query>
    <ap:Resource>
      <ap:Name>Clint Hill</ap:Name>
      <ap:Guid>5B423370-D161-43F6-AE72-D91023322236</ap:Guid>
    </ap:Resource>
  </ap:Resources>
</ap:Requests>
```

If *chkReturnDetail* = true, an <ap:ResourceDetail> element is returned, which contains details about the resource.

For example:

```
<ap:Requests>
   <ap:Resources Id="ApiSample.listQueryResources">
      <ap:Query>resource(name="Clint Hill")</ap:Query>
     </ap:ResourceDetail>
             .
             .
             .
```

For information about the elements contained in the <ap:ResourceDetail> element, see getResourceDetail.

Miscellaneous Requests

Server requests are available for miscellanous tasks.

Ping

This request verifies the server is responsive.

Syntax

```
com.tibco.wcc.base.Requests.ping(requestId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

The **ping** request returns the following:

```
<ap:Requests>
   <ap:Ping Id="ApiSample.ping" />
</ap:Requests>
```

workItemFilterSortFields

This request returns a list of fields that can be used to sort or filter a work item list.

Syntax

```
com.tibco.wcc.base.Requests.workItemFilterSortFields(requestId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

Returns an <ap:Field> element for each filterable and sortable field. The <ap:Field> element contains the following elements:

- <ap:ID> - Identifies the field in this list. The ID starts at 1 and increases by 1 for each field in the list returned by the **workItemFilterSortFields** request.
- <ap:Name> - The name of the field.
- <ap:Description> - The description of the field.
- <ap:Type> - Describes the type of data in the field: COL_STRING, COL_NUMERIC, COL_DATETIME, COL_BOOLEAN, COL_STATE, or COL_DISTSTRATEGY.
- <ap:IsSortField> - true = field is sortable; false = field is not sortable.
- <ap:IsFilterField> - true = field is filterable; false = field is not filterable.

For example:

```
<ap:Requests>
   <ap:Fields Id="ApiSample.workItemFilterSortFields">
      <ap:Field>
         <ap:ID>1</ap:ID>
         <ap:Name>id</ap:Name>
         <ap:Description>The integer value that denotes the ID of the work
item.          </ap:Description>
         <ap:Type>COL_NUMERIC</ap:Type>
         <ap:IsSortField>true</ap:IsSortField>
         <ap:IsFilterField>true</ap:IsFilterField>
      </ap:Field>
              .
              .
              .
   </ap:Fields>
</ap:Requests>
```

getResourceOrderFilterCriteria

This request returns the default sort and filter criteria that are defined on the server for the currently logged on user. These criteria are used to display work item lists for a user before a custom filter and sort are specified in the application.

The default sort and filter criteria can be set in a number of ways:

- AMX BPM API calls - There are API calls that allow the sort and filter criteria on the server to be set for a resource. For information, see the *TIBCO ActiveMatrix BPM Developer's Guide*.
- Use the **setResourceOrderFilterCriteria** request - See setResourceOrderFilterCriteria.
- When the **Open Next Work Item** function is used. This function always uses the sort and filter criteria on the server to determine the next work item in the list. Therefore, whenever you select the **Open Next Work Item** function, a comparison is made between the currently defined sort and filter in the work item list, and the sort and filter specified on the server. If they are the same, the function uses that definition to determine the next work item. If they differ, the sort and filter specified in the current work item list is written to the server, resulting in a *new* default sort and filter defined on the server. The **Open Next Work Item** function then uses that sort and filter to determine the next work item to open.

Syntax

```
com.tibco.wcc.base.Requests.getResourceOrderFilterCriteria(requestId);
```

Parameters

requestId - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

Returns an <ap:OrderFilterCriteria> element, which contains the following elements:

- <ap:Filter> - The default filter expression stored on the server.
- <ap:Sort> - The default sort expression stored on the server.

For example:

```
<ap:Requests>
  <ap:OrderFilterCriteria Id="ApiSample.getResourceOrderFilterCriteria">
    <ap:Filter>startDate &lt; 2011-03-04T00:00:00.000-08:00</ap:Filter>
    <ap:Sort>startDate ASC, priority DESC</ap:Sort>
  </ap:OrderFilterCriteria>
</ap:Requests>
```

setResourceOrderFilterCriteria

This request sets the default sort and filter criteria on the server for the currently logged on user.

For more information, see getResourceOrderFilterCriteria.

Syntax

```
com.tibco.wcc.base.Requests.setResourceOrderFilterCriteria(requestId,
                                                           txtFilter,
                                                           txtSort);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *txtFilter* - (String) The new filter expression to write to the server. For information about filter syntax, see the "Sorting and Filtering Work Item Lists" topic in the *TIBCO ActiveMatrix BPM Developer's Guide*.

  For example:

  ```
        startDate < 2011-03-11T00:00:00.000-08:00
  ```

- *txtSort* - (String) The new sort order to apply to write to the server. Specify the sort field, following by either ASC or DESC, for ascending or descending, respectively. Separate multiple sort fields with commas. For example:

  ```
        priority ASC, startDate DESC
  ```

  Use the following sort field names in the *txtSort* parameter:

Attribute	Sort field name
Attribute #1 - Attribute #40	attribute1 - attribute40
Distribution Strategy	distributionStrategy
Target Date	endDate
Instance Description	appInstanceDescription
ID	appInstance
Priority	priority
Process Template	appName
Start Date	startDate
Work Item ID	id

Returns

No return, other than the request ID (and the return status information):

```
<ap:Requests>
    <ap:OrderFilterCriteria Id="ApiSample.setResourceOrderFilterCriteria">
    </ap:OrderFilterCriteria>
</ap:Requests>
```

getUserPrivileges

This request returns a list of the privileges held by the specified user.

Syntax

```
com.tibco.wcc.base.Requests.getUserPrivileges(requestId,
                                              txtResourceGuid);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *txtResourceGuid* - (String) The GUID for the resource whose privileges are to be returned.

Returns

Returns an <ap:XmlQualifiedModelEntity> element for each privilege held by the resource. The <ap:XmlQualifiedModelEntity> element contains the following elements:

- <ap:Name> - The name of the privilege.

- <ap:Qualifier> - This is not applicable in this context, and is explicitly set to an empty string internally
- <ap:Guid> - The GUID for the privilege.
- <ap:ModelVersion> - The version number of the organization model. A -1 signifies the latest version.
- <ap:EntityType> - Always PRIVILEGE in this context.

For example:

```
<ap:Requests>
   <ap:GetUserPrivilegesResponse Id="ApiSample.getUserPrivileges">
     <ap:UserGuid>5B423370-D161-43F6-AE72-D91023322236</ap:UserGuid>
        <ap:XmlQualifiedModelEntities>
           <ap:XmlQualifiedModelEntity>
              <ap:Name>ResourceManager</ap:Name>
              <ap:Qualifier />
              <ap:Guid>_UNb_kMpSEd64gM7QE8RwxA</ap:Guid>
              <ap:ModelVersion>-1</ap:ModelVersion>
              <ap:EntityType>PRIVILEGE</ap:EntityType>
           </ap:XmlQualifiedModelEntity>
                        .
                        .
                        .
        </ap:XmlQualifiedModelEntities>
   </ap:GetUserPrivilegesResponse>
</ap:Requests>
```

listSystemActionAuthorizedEntities

This request returns a list of GUIDs identifying the organizational entities that are granted the specified system action.

Syntax

```
com.tibco.wcc.base.Requests.listActionAuthorisedEntities(requestId,
                                                         systemAction);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *systemAction* - (String) The system action. Note that you can only specify the *scoped* system actions, that is, those that can be set at the group, organization unit, or position level. They are:

  - BRM.openOtherResourcesItems

  - BRM.closeOtherResourcesItems

  - BRM.reallocateWorkItemToWorld

  - BRM.reallocateToOfferSet

  - BRM-viewWorkList

  - BRM.workItemAllocation

  - BRM.setResourceOrderFilterCriteria

Returns

Returns a <ap:Guid> element for each organizational entity that is assigned the specified scoped system action. The <ap:Guid> element contains the GUID of the organizational entity to which the system action is assigned.

For example:

```
<ap:Requests>
  <ap:ListActionAuthorisedEntitiesResponse
```

```
Id="ApiSample.listActionAuthorisedEntities">
    <ap:Guids>
      <ap:Guid>_9y7hYMpREd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_O2JmsFntEd-wEpVSqlaWLw</ap:Guid>
      <ap:Guid>_ArjUUMpSEd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_CWsngMpSEd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_6lTHwMpREd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_8NnJYMpREd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_0Ogk0MpREd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_4oI3gMpREd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>__O368MpREd64gM7QE8RwxA</ap:Guid>
      <ap:Guid>_2ieLgMpREd64gM7QE8RwxA</ap:Guid>
    </ap:Guids>
  </ap:ListActionAuthorisedEntitiesResponse>
</ap:Requests>
```

isActionAuthorized

This request returns true or false, indicating whether the specified system action is authorized for the currently logged-in user. For scoped system actions, you can also specify an organizational entity.

For general information about system actions, see the "Configuring User Access" topic in the *TIBCO Workspace Configuration and Customization* guide.

For a list of available system actions, also see the User Access Privileges dialog in the Workspace application (**Help** > **User Access Privileges**).

Syntax

```
com.tibco.wcc.base.Requests.isActionAuthorised(requestId,
                                               systemActions,
                                               scope);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *systemActions* - (Array[String]) The system actions for which authorization is being checked.
- *scope* (Object) (optional) This specifies an organizational entity to determine if a scoped system action is authorized:

  ```
  {
  entityType: entityType,
  guid: guid,
  modelVersion: modelVersion,
  qualifier: qualifier
  };
  ```

  where:

- *entityType* - (String) The organizational entity: RESOURCE, GROUP, POSITION, ORGANIZATION_UNIT, or ORGANIZATION.
- *guid* (String) The GUID for the organizational entity.
- *ModelVersion* (String) Version of the organization model containing the organizational entity.
- *qualifier* (String) (optional) Not used — set to empty string internally.

Returns

Returns an <ap:Action> element, which contains the following elements:

- <ap:Component> - The component to which the system action is associated.
- <ap:Name> - The name of the system action.

- <ap:Authorized> - true = the system action is authorized; false = the system action is not authorized.

For example:

```
<ap:Requests>
  <ap:IsActionAuthorisedResponse Id="ApiSample.isActionAuthorized">
    <ap:Actions>
      <ap:Action>
        <ap:Component>DE</ap:Component>
        <ap:Name>userAdmin</ap:Name>
        <ap:Authorised>true</ap:Authorised>
      </ap:Action>
    </ap:Actions>
  </ap:IsActionAuthorisedResponse>
</ap:Requests>
```

Work Item Requests

Server requests are available for working with work items.

listWorkItems

This request returns a list of the work items for the logged-in user.

Syntax

```
com.tibco.wcc.base.Requests.listWorkItems(requestId,

numStartIndex,
                                          numReturnCount,
                                          txtFilter,
                                          txtSort);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *numStartIndex* - (Number) Index of the first item to return.
- *numReturnCount* - (Number) The number of work items to return.
- *txtFilter* - (String) The filter expression to apply to determine the work items to return. For information about filter syntax, see the "Sorting and Filtering Work Item Lists" topic in the developer's guide for your BPM runtime environment.

  For example:

  ```
  startDate < 2011-03-11T00:00:00.000-08:00
  ```

- *txtSort* - (String) The sort order to apply to the work items returned. Specify the sort field, following by either ASC or DESC, for ascending or descending, respectively. Separate multiple sort fields with commas. For example:

  ```
  priority ASC, startDate DESC
  ```

  Use the following sort field names in the *txtSort* parameter:

Attribute	Sort field name
Attribute #1 - Attribute #40	attribute1 - attribute40
Distribution Strategy	distributionStrategy
Target Date	endDate

Attribute	Sort field name
Instance Description	appInstanceDescription
ID	appInstance
Priority	priority
Process Template	appName
Start Date	startDate
Work Item ID	id

Returns

This request returns the following elements:

- <ap:StartPosition> - The index number of the first work item returned.
- <ap:EndPosition> - The index number of the last work item returned.
- <ap:TotalItems> - The total number of work items returned.
- <ap:WorkItems> - This contains an <ap:WorkItem> element for each work item returned. The <ap:WorkItem> element contains the following elements:
  - <ap:ID> - The work item ID.
  - <ap:Version> - The number of times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by 1 each time it changes state.
  - <ap:Name> - The name of the task in the process to which the work item is associated.
  - <ap:Description> - The description of the task in the process to which the work item is associated.
  - <ap:StartDate> - Date and time the work item was created and arrived in the work item list.
  - <ap:EndDate> - A deadline date and time for the work item. Note that this is shown as “Target Date” by default in the application.
  - <ap:DistributionStrategy> - Method used to distribute the work item when it was originally created, either “Offer” or “Allocate”.
  - <ap:Priority> - Numeric value indicating the relative importance of the work item.
  - <ap:GroupID> - Identifies a number of work items that are grouped together in the process definition for some purpose. A 0 (zero) indicates the work item is not grouped with others.
  - <ap:ActivityID> - Identifies the user activity within the process that generated the work item. This number is used in the event viewer to identify the work item across all components.
  - <ap:ActivityName> - The name of the user task that generated the work item.
  - <ap:AppID> - A GUID for the application.
  - <ap:AppInstance> - The ID of the process instance from which the work item was created.
  - <ap:AppInstanceDescription> - The description of the process instance from which the work item was created.
  - <ap:AppName> - Name of process that was started to create this work item.

- <ap:ScheduleStatus> - Indicates if the work item is inside its schedule period, which is considered from the start date/time to the target date/time. The possible values are: BEFORE, DURING, AFTER and NO_SCHEDULE.
- <ap:State> - The work item's current state: OFFERED, ALLOCATED, CREATED, OPENED, PENDED, PENDHIDDEN, or SUSPENDED.
- <ap:WorkTypeID> - The work type ID.
- <ap:WorkTypeUID> - The work type universal ID.
- <ap:WorkTypeVersion> - The work type version number.
- <ap:WorkTypeDescription> - The work type description.
- <ap:Visible> - true = work item is visible; false = work item is hidden.
- <ap:Attributes> - This element contains sub-elements <ap:Attribute1> through <ap:Attribute40> representing Attributes 1-40 in the work item, but only for those attributes that contain a value.

  For example:

```
<ap:Requests>
  <ap:ListWorkItems Id="ApiSample.listWorkItems">
    <ap:StartPosition>0</ap:StartPosition>
    <ap:EndPosition>3</ap:EndPosition>
    <ap:TotalItems>4</ap:TotalItems>
    <ap:WorkItems>
      <ap:WorkItem>
        <ap:ID>52</ap:ID>
        <ap:Version>2</ap:Version>
        <ap:Name>Respond</ap:Name>
        <ap:Description>Respond</ap:Description>
        <ap:StartDate>2011-02-25T08:54:53.657-08:00</ap:StartDate>
        <ap:EndDate />
        <ap:DistributionStrategy>OFFER</ap:DistributionStrategy>
        <ap:Priority>50</ap:Priority>
        <ap:GroupID>0</ap:GroupID>
        <ap:ActivityID>pvm:001i2h</ap:ActivityID>
        <ap:ActivityName>Respond</ap:ActivityName>
        <ap:AppID>_Y_SqoV6zEd-KqJmYNEw8ow</ap:AppID>
        <ap:AppInstance>pvm:0a129</ap:AppInstance>
        <ap:AppInstanceDescription />
        <ap:AppName>CSCallbackProcess</ap:AppName>
        <ap:ScheduleStatus>DURING</ap:ScheduleStatus>
        <ap:State>OFFERED</ap:State>
        <ap:WorkTypeID />
        <ap:WorkTypeUID />
        <ap:WorkTypeVersion />
        <ap:WorkTypeDescription />
        <ap:Visible>true</ap:Visible>
        <ap:Attributes>
          <ap:Attribute1>Retired</ap:Attribute1>
          <ap:Attribute2>2010</ap:Attribute2>
        </ap:Attributes>
      </ap:WorkItem>
             .
             .
             .
    </ap:WorkItems>
  </ap:ListWorkItems>
</ap:Requests>
```

listSupervisedWorkItems

The request returns a list of work items for another resource or for an organizational entity (group, position, organizational unit, or organization).

Syntax

```
com.tibco.wcc.base.Requests.listSupervisedWorkItems(requestId,
                                            numStartIndex,
                                            numReturnCount,
                                            txtFilter,
                                            txtSort,
                                            txtEntityType,
                                            txtEntityGuid,
                                            selSupervisedWorkItemState,
                                            txtEntityModelVersion,
                                            txtEntityQualifier);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *numStartIndex* - (Number) Index of the first item to return.
- *numReturnCount* - (Number) The number of work items to return.
- *txtFilter* - (String) The filter expression to apply to determine the work items to return. For information about filter syntax, see the "Sorting and Filtering Work Item Lists" topic in the developer's guide for your BPM runtime environment.

  For example:

  ```
  startDate < 2011-03-11T00:00:00.000-08:00
  ```
- *txtSort* - (String) The sort order to apply to the work items returned. Specify the sort field, following by either ASC or DESC, for ascending or descending, respectively. Separate multiple sort fields with commas. For example:

  ```
  priority ASC, startDate DESC
  ```

  Use the following sort field names in the *txtSort* parameter:

Attribute	Sort field name
Attribute #1 - Attribute #40	attribute1 - attribute40
Distribution Strategy	distributionStrategy
Target Date	endDate
Instance Description	appInstanceDescription
ID	appInstance
Priority	priority
Process Template	appName
Start Date	startDate
Work Item ID	id
- *txtEntityType* - (String) Identifies the type of supervised organizational entity (RESOURCE, GROUP, POSITION, ORGANIZATION_UNIT or ORGANIZATION).
- *txtEntityGuid* - (String) The GUID for the organizational entity.
- *selSupervisedWorkItemState* - (String) - Identifies the state of the work items that are to be returned: 'Allocated' or 'Offered'.
- *txtEntityModelVersion* - (String) The version of the organizational model that contains the organizational entity — not applicable to the RESOURCE entity type. Passing -1 causes it to use the most recent version.

- *txtEntityQualifier* - (String) This is not currently used.

Returns

This request returns the same elements as the **listWorkItems** request — for information, see listWorkItems.

For example:

```
<ap:Requests>
  <ap:ListWorkItems Id="ApiSample.listSupervisedWorkItems">
    <ap:StartPosition>0</ap:StartPosition>
    <ap:EndPosition>3</ap:EndPosition>
    <ap:TotalItems>4</ap:TotalItems>
    <ap:WorkItems>
      <ap:WorkItem>
        <ap:ID>7</ap:ID>
        <ap:Version>10</ap:Version>
        <ap:Name>ConfirmResolution</ap:Name>
        <ap:Description>Confirm Resolution</ap:Description>
        <ap:StartDate>2011-02-23T17:08:11.780-08:00</ap:StartDate>
        <ap:EndDate />
        <ap:DistributionStrategy>OFFER</ap:DistributionStrategy>
        <ap:Priority>50</ap:Priority>
        <ap:GroupID>0</ap:GroupID>
        <ap:ActivityID>pvm:001iv</ap:ActivityID>
        <ap:ActivityName>ConfirmResolution</ap:ActivityName>
        <ap:AppID>_-rTukV60Ed-KqJmYNEw8ow</ap:AppID>
        <ap:AppInstance>pvm:0a123</ap:AppInstance>
        <ap:AppInstanceDescription />
        <ap:AppName>HelpDeskProcess</ap:AppName>
        <ap:ScheduleStatus>DURING</ap:ScheduleStatus>
        <ap:State>OFFERED</ap:State>
        <ap:WorkTypeID />
        <ap:WorkTypeUID />
        <ap:WorkTypeVersion />
        <ap:WorkTypeDescription />
        <ap:Visible>true</ap:Visible>
        <ap:Attributes />
      </ap:WorkItem>
               .
               .
               .
    </ap:WorkItems>
  </ap:ListWorkItems>
</ap:Requests>
```

cancelWorkItems

This returns opened work items to an unopened state.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see cancelWorkItem.

skipWorkItems

This marks the specified work items as complete, removes them from the work item list, and causes the process to advance. It has the same affect as opening the work items and submitting them.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see skipWorkItem.

reofferWorkItems

This re-offers the specified work items.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see reofferWorkItem.

getOfferSet

This request returns a list of resources that were offered a specified work item.

Syntax

```
com.tibco.wcc.base.Requests.getOfferSet(requestId,
                                        workItemId,
                                        workItemVersion);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *workItemId* - (String) The work item whose offer set is returned.
- *workItemVersion* - (String) (optional) The number of times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by 1 each time it changes state.

Returns

Returns an <ap:ResourceGuids> element, which contains an <ap:ResourceGuid> element for each resource in the offer set.

For example:

```
<ap:Requests>
  <ap:ResourceGuids Id="ApiSample.getOfferSet_0">
    <ap:ResourceGuid>tibco-admin</ap:ResourceGuid>
    <ap:ResourceGuid>5B423370-D161-43F6-AE72-D91023322236</ap:ResourceGuid>
    <ap:ResourceGuid>4945E03D-7EFC-4F8C-831C-497F28CAD0C8</ap:ResourceGuid>
    <ap:ResourceGuid>BF497153-81E3-4214-B7CB-8EB1C8137186</ap:ResourceGuid>
    <ap:ResourceGuid>191724ED-711C-4D0C-8638-E677E55F7616</ap:ResourceGuid>
    <ap:ResourceGuid>E3A3FC77-10FD-4326-9F56-FE230C6FC579</ap:ResourceGuid>
    <ap:ResourceGuid>E2B915CD-476E-4291-990D-7E507982773C</ap:ResourceGuid>
    <ap:ResourceGuid>ADFF6416-C0E6-4FD6-A54F-9BDE3731412C</ap:ResourceGuid>
    <ap:ResourceGuid>4E55BCCF-62C5-4D1D-85CA-2C8C7E743DCE</ap:ResourceGuid>
  </ap:ResourceGuids>
</ap:Requests>
```

allocateWorkItems

This request allocates the specified work items to the specified resource.

Syntax

```
com.tibco.wcc.base.Requests.allocateWorkItems(requestId,
                                              arrayWorkItems,
                                              txtResourceGuid);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- arrayWorkItems - (Array[Object]) The work items to be allocated.

  ```
  [
    {
       workItemId: workItemId,
       workItemVersion: workItemVersion,
    }
  ]
  ```

  where:
- *workItemId* - (String) Unique ID for the work item.
- *workItemVersion* - (String) (optional) The number of times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by 1 each time it changes state.
- txtResourceGuid - (String) The GUID for the resource to which the work items are to be allocated.

Returns

Returns an <ap:WorkItem> element for each work item that is allocated.

For example:

```
<ap:Requests>
  <ap:WorkItems Id="ApiSample.allocateWorkItems">
    <ap:WorkItem>
           .
           .
           .
    </ap:WorkItem>
           .
           .
           .
  </ap:WorkItems>
</ap:Requests>
```

For details about the contents of the <ap:WorkItem> element, see listWorkItems.

reallocateWorkItems

This request takes allocated work items and changes the allocation to another resource, optionally cancelling changes made by the original resource.

Syntax

```
com.tibco.wcc.base.Requests.reallocateWorkItems(requestId,
                                                arrayWorkItems,
                                                txtResourceGuid,
                                                chkRevertData);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *arrayWorkItems* - (Array[Object]) The work items to be re-allocated.

  ```
  [
    {
       workItemId: workItemId,
       workItemVersion: workItemVersion,
  ```

```
  }
]
```

where:

- *workItemId* - (String) Unique ID for the work item.
- *workItemVersion* - (String) (optional) The number of times the work item has changed state. The version number starts at 0 when the work item is created, and is incremented by 1 each time it changes state.

- *txtResourceGuid* - (String) The GUID for the resource to which the work items are to be re-allocated.
- *chkRevertData* - (Boolean) Specifies whether or not pending changes inside the work items, that were made by the original resource, will be preserved when the work items are re-allocated.

Returns

Returns an <ap:WorkItem> element for each work item that is re-allocated.

For example:

```
<ap:Requests>
  <ap:WorkItems Id="ApiSample.reallocateWorkItems">
    <ap:WorkItem>
           .
           .
           .
    </ap:WorkItem>
           .
           .
           .
  </ap:WorkItems>
</ap:Requests>
```

For details about the contents of the <ap:WorkItem> element, see listWorkItems.

pendWorkItems

This hides allocated work items until a specified date / time is reached or until a specified period of time has elapsed.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see pendWorkItem.

Process Template and Instance Requests

Server requests are available for working with process templates and instances.

listStartTemplates

This request returns a list of process templates that require no input parameters, so they can be started directly without having to invoke a business service that might include forms.

Syntax

```
com.tibco.wcc.base.Requests.listStartTemplates(requestId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

Returns an <ap:QualifiedProcessName> element for each process template that can be started directly. The <ap:QualifiedProcessName> element contains the following elements:

- <ap:ModuleName> - The path to the XPDL file that defines the "process package".
- <ap:ProcessName> - The name of the process template.
- <ap:Version> - The version of the process template.

For example:

```
<ap:Requests>
  <ap:ListStartTemplates Id="ApiSample.listStartTemplates">
    <ap:QualifiedProcessNames>
      <ap:QualifiedProcessName>
        <ap:ModuleName>/HelpDesk/Process Packages/HelpDesk.xpdl</ap:ModuleName>
        <ap:ProcessName>InternalHelpDesk</ap:ProcessName>
        <ap:Version>1.0.0.201101101430</ap:Version>
      </ap:QualifiedProcessName>
                    .
                    .
                    .
    </ap:QualifiedProcessNames>
  </ap:ListStartTemplates>
</ap:Requests>
```

startProcessInstance

This starts an instance of a process template that requires no input parameters.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see startProcessInstance.

queryProcessTemplateCount

This request returns the number of process templates that exist. You can also limit the templates counted by passing a filter expression that the templates must satisfy.

Syntax

```
com.tibco.wcc.base.Requests.queryProcessTemplateCount(requestId,
                                                      txtFilter);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *txtFilter* - (String) A filter expression to limit the number of process templates. Only the process templates that satisfy the expression are counted. For information about filter syntax, see the "Sorting and Filtering Process Lists" topic in the developer's guide for your BPM runtime environment (note that you only need to provide the *condition* part of the WHERE clause that is documented in the "Sorting and Filtering Process Lists" topic).

  For a list of the attributes that can be used in the filter expression, use the listProcessTemplateAttributes request.

Returns

Returns an <ap:QueryProcessTemplateCount> element that contains the process template count.

For example:

```
<ap:Requests>
    <ap:QueryProcessTemplateCount Id="ApiSample.queryProcessTemplateCount">17
    </ap:QueryProcessTemplateCount>
</ap:Requests>
```

listProcessTemplateAttributes

This request returns a list of attributes that can be used for filtering or sorting process templates.

Syntax

```
com.tibco.wcc.base.Requests.listProcessTemplateAttributes(requestId,
                                                          selType);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *selType* - (Number) Specifies whether filter attributes or sort attributes are to be returned. 1 = return filter attributes; 2 = return sort attributes.

Returns

Returns an <ap:TemplateAttribute> element for each filter or sort attribute. The <ap:TemplateAttribute> element contains the following elements:

- <ap:Name> - The name of the filter or sort attribute.
- <ap:Type> - The type of data stored in the attribute.

For example:

```
<ap:Requests>
  <ap:TemplateAttributes Id="ApiSample.listProcessTemplateAttributes">
    <ap:TemplateAttribute>
      <ap:Name>DEFINITION.PRIORITY</ap:Name>
      <ap:Type>short</ap:Type>
    </ap:TemplateAttribute>
                 .
                 .
                 .
   </ap:TemplateAttributes>
</ap:Requests>
```

queryProcessTemplates

This request returns a list of process templates.

Syntax

```
com.tibco.wcc.base.Requests.queryProcessTemplates(requestId,
                                                  txtFilter,
                                                  txtSort);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *txtFilter* - (String) A filter expression to limit the number of process templates. Only the process templates that satisfy the expression are returned. For information about filter syntax, see the "Sorting and Filtering Process Lists" topic in the developer's guide for your BPM runtime environment.

For a list of the attributes that can be used in the filter expression, use the listProcessTemplateAttributes request.

- *txtSort* - (String) The sort order to apply to the process templates returned. Specify the sort attribute, following by either ASC or DESC, for ascending or descending, respectively. Separate multiple sort attributes with commas. For example:

  ```
  DEFINITION.PRIORITY ASC, DEFINITION.NAME DESC
  ```

  For a list of the attributes that can be used in the filter expression, use the listProcessTemplateAttributes request.

Returns

Returns an <ap:ProcessTemplate> element for each process template returned. The <ap:ProcessTemplate> element contains the following elements:

- <ap:DEFINITION.DESCRIPTION> - Description of the process template.
- <ap:DEFINITION.VERSION> - Version of the process template.
- <ap:DEFINITION.NAME> - Name of the process template.
- <ap:MODULE.NAME> - The path to the XPDL file that defines the "process package".
- <ap:DEFINITION.PRIORITY> - The priority of the process template.
- <ap:DEFINITION.CREATION_DATE> - The date/time the process template was created.

For example:

```
<ap:Requests>
  <ap:QueryProcessTemplates Id="ApiSample.queryProcessTemplates">
    <ap:QueryId>0</ap:QueryId>
    <ap:ProcessTemplateCount>16</ap:ProcessTemplateCount>
    <ap:ProcessTemplates>
      <ap:ProcessTemplate>
        <ap:DEFINITION.DESCRIPTION>*** Generated by TIBCO Business Studio.</
ap:DEFINITION.DESCRIPTION>
        <ap:DEFINITION.VERSION>1.0.0.201101101413</ap:DEFINITION.VERSION>
        <ap:DEFINITION.NAME>Fields3Process</ap:DEFINITION.NAME>
        <ap:MODULE.NAME>/Acme-CommonFields/Process Packages/CommonFields.xpdl</
ap:MODULE.NAME>
        <ap:DEFINITION.PRIORITY>NORMAL</ap:DEFINITION.PRIORITY>
        <ap:DEFINITION.CREATION_DATE>2011-02-23T16:31:02.390-08:00</
ap:DEFINITION.CREATION_DATE>
    </ap:ProcessTemplate>
                .
                .
                .
    </ap:ProcessTemplates>
  </ap:QueryProcessTemplates>
</ap:Requests>
```

listProcessInstanceAttributes

This request returns the attributes associated with instances of a particular process template.

Syntax

```
com.tibco.wcc.base.Requests.listProcessInstanceAttributes(requestId,
                                                     txtProcessName,
                                                     txtModuleName,
                                                     txtVersion,
                                                     selType);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

- *txtProcessName* - (String) The name of the process template whose instance's attributes are to be returned.
- *txtModuleName* - (String) The path to the XPDL file that defines the "process package".
- *txtVersion* - (String) The version of the process template.
- *selType* - (Number) Specifies the type of attributes to return. 1 = filter attributes; 2 = sort attributes; 3= both filter and sort attributes.

Returns

Returns an <ap:InstanceAttribute> element for each process instance attribute returned. The <ap:InstanceAttribute> element contains the following elements:

- <ap:ModuleName> - The path to the XPDL file that defines the "process package".
- <ap:ProcessName> - The name of the process template.
- <ap:Version> - The version of the process template.
- <ap:Name> - The name of the attribute.
- <ap:Type> - The type of data stored in the attribute.

For example:

```
<ap:Requests>
  <ap:InstanceAttributes Id="ApiSample.listProcessInstanceAttributes">
    <ap:InstanceAttribute>
      <ap:ModuleName>/CSCallback/Process Packages/CSCallback.xpdl</ap:ModuleName>
      <ap:ProcessName>CSCallbackProcess</ap:ProcessName>
      <ap:Version>1.0.0.201101101507</ap:Version>
      <ap:Name>ResolutionDetails</ap:Name>
      <ap:Type>string</ap:Type>
    </ap:InstanceAttribute>
                 .
                 .
                 .
  </ap:InstanceAttributes>
</ap:Requests>
```

queryProcessInstanceCount

This request returns the number of instances that exist for the specified process template(s).

Syntax

```
com.tibco.wcc.base.Requests.queryProcessInstanceCount(requestId,
                                                      arrProcessTemplates,
                                                      txtFilter);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- arrayProcessTemplates - (Array[Object]) The process templates whose instances are to be counted.

  ```
  [
    {
       moduleName: moduleName,
       processName: processName,
       processVersion : processVersion
    }
  ]
  ```

  where:
- *moduleName* - (String) The path to the XPDL file that defines the "process package".

- *processName* - (String) The name of the process template whose instances are to be counted.
- *processVersion* - (String) The version of the process template.
- *txtFilter* - (String) A filter expression to limit the number of process instances. Only the process instances that satisfy the expression are counted. For information about filter syntax, see the "Sorting and Filtering Process Lists" topic in the developer's guide for your BPM runtime environment.

  For a list of the attributes that can be used in the filter expression, use the listProcessInstanceAttributes request.

Returns

Returns an <ap:QueryProcessInstanceCount> element, which contains the process instance count.

For example:

```
<ap:Requests>
   <ap:QueryProcessInstanceCount Id="ApiSample.queryProcessInstanceCount">5</
ap:QueryProcessInstanceCount>
</ap:Requests>
```

queryProcessInstances

This request returns the instances for the specified process templates.

Syntax

```
com.tibco.wcc.base.Requests.queryProcessInstances(requestId,
                                              arrProcessTemplates,
                                              txtFilter,
                                              txtSort,
                                              numPageSize,
                                              arrAttributes,
                                              arrAttributeTypes);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- arrayProcessTemplates - (Array[Object]) The process templates whose instances are to be returned.

  ```
  [
    {
       moduleName: moduleName,
       processName: processName,
       processVersion : processVersion
    }
  ]
  ```

  where:

- *moduleName* - (String) The path to the XPDL file that defines the “process package”.
- *processName* - (String) The name of the process template whose instances are to be returned.
- *processVersion* - (String) The version of the process template.

  Note that if you specify *any* of the next three parameters — *txtFilter*, *txtSort*, *numPageSize* — you *must* include attributes (*arrAttributes*) in the method call. If you do not specify any of the next three parameters, including attributes is optional. If you do not specify attributes, they are all returned.

- *txtFilter* - (String) A filter expression to limit the number of process instances. Only the process instances that satisfy the expression are returned. (Also see note above.) For information about filter syntax, see the "Sorting and Filtering Process Lists" topic in the developer's guide for your BPM runtime environment.

  For a list of the attributes that can be used in the filter expression, use the listProcessInstanceAttributes request.

- *txtSort* - (String) The sort order to apply to the process instances returned. Specify the sort attribute, following by either ASC or DESC, for ascending or descending, respectively. Separate multiple sort attributes with commas. (Also see note above.) For example:

  ```
  INSTANCE.PRIORITY ASC, INSTANCE.NAME DESC
  ```

  For a list of the attributes that can be used in the sort order, use the listProcessInstanceAttributes request.

- *numPageSize* - (Number) The number of process instances per page to return. Specify 0 (zero) to return all process instances on a single page. (Also see note above.)

- *arrAttributes* - (Array[String]) The attributes to return with the process instances. Also see the note above — if you do not specify attributes, they are all returned.

  Note that the attributes you specify *must exist in all of the specified process templates*. There are a number of "system attributes" (for example, INSTANCE.START_DATE, INSTANCE.PRIORITY, etc.) that exist for all templates, but you must be careful that attributes associated with the process instance data are only requested if attributes of the same name are in all of the specified templates. If a requested attribute is not valid for a process template, instances of that template are not returned in the response.

  Also note that if you include any attributes in this parameter, at least one of them *must* be INSTANCE.VERSION, INSTANCE.NAME, or MODULE.NAME.

- *arrAttributeTypes* - (String) An array of attribute names and data types. For each attribute that is referenced in a sort or filter expression, you must include that attribute in this array.

Returns

Returns an <ap:QueryProcessInstances> element, which contains the following elements:

- <ap:QueryId> - If 0 (zero) was specified in the *numPageSize* parameter, the value in this element is 0 (zero). If you specified a page size other than 0, this element contains an ID that can be used as the input to the queryProcessInstancesNextPage, queryProcessInstancesPreviousPage, and queryDone requests.

- <ap:ProcessInstanceCount> - The number of process instances returned. If you have requested paging (the *arrPageSize* parameter contained a value other than 0), this is the number returned on the current page.

- <ap:ProcessInstances> - Contains an <ap:ProcessInstance> element for each process instance that is returned. The <ap:ProcessInstance> element contains the following elements:

  – <ap:INSTANCE.VERSION> - The version of the process instance.

  – <ap:INSTANCE.NAME> - The name of the process instance.

  – <ap:INSTANCE.ID> - An identifier for the process instance.

  – <ap:MODULE.NAME> - The path to the XPDL file that defines the "process package".

  – <ap:INSTANCE.STATUS> - The status of the instance: Active, Canceling, Completing, Failing, Halted, Halting, Not started, Restarting, Resuming, Starting, Suspended, or Suspending. For descriptions of all of these statuses, see the *TIBCO Workspace User's Guide*.

  – <ap:INSTANCE.WAITING_WORK_COUNT> - The number of outstanding work items in the process instance.

  – <ap:INSTANCE.PRIORITY> - The priority value of the process instance.

  – <ap:INSTANCE.PARENT_PROCESS_ID> - The instance ID of the parent process instance — only applicable for re-usable sub-processes.

  – <ap:INSTANCE.START_DATE> - The date and time the process instance was started.

Each of these elements represents a “system attribute”. Note that all system attributes are returned in the response, however, only those that you requested in the **queryProcessInstances** request are populated with values.

If you have specified attributes that contain instance-specific data (for example <ap:ContactPhone>, <ap:IssueDescription>, etc.) in the request, those are also returned under the <ap:ProcessInstance> element.

For example:

```
<ap:Requests>
  <ap:QueryProcessInstances Id="ApiSample.queryProcessInstances">
    <ap:QueryId>-1909332857</ap:QueryId>
    <ap:ProcessInstanceCount>3</ap:ProcessInstanceCount>
    <ap:ProcessInstance>
      <ap:INSTANCE.VERSION>1.0.0.201101101507</ap:INSTANCE.VERSION>
      <ap:INSTANCE.NAME/>
      <ap:INSTANCE.ID>pvm:0a122</ap:INSTANCE.ID>
      <ap:MODULE.NAME/>
      <ap:INSTANCE.STATUS/>
      <ap:INSTANCE.WAITING_WORK_COUNT>0</ap:INSTANCE.WAITING_WORK_COUNT>
      <ap:INSTANCE.PRIORITY>0</ap:INSTANCE.PRIORITY>
      <ap:INSTANCE.START_DATE>2011-02-23T17:07:00.767-08:00</ap:INSTANCE.START_DATE>
      <ap:ContactPhone>509-555-1212</ap:ContactPhone>
      <ap:IssueDescription>He wants more information.</ap:IssueDescription>
    </ap:ProcessInstance>
               .
               .
               .
    </ap:ProcessInstances>
  </ap:QueryProcessInstances>
</ap:Requests>
```

queryProcessInstancesNextPage

This request returns the next page of process instances — used after calling the queryProcessInstances request with the page size set to a value other than 0. You must pass in a query ID that was returned in the response from the **queryProcessInstances** request.

Note that if you call this request, and there are no more process instances to return, an exception is thrown. (Also see the queryProcessInstanceCount request for a total count.)

Syntax

```
com.tibco.wcc.base.Requests.queryProcessInstancesNextPage(requestId,
                                                          queryId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *queryId* - (String) A unique identifier for the **queryProcessInstances** request in which you have requested paging for the returned process instances. The value you must pass in this parameter is returned in the <ap:QueryId> element in the **queryProcessInstances** request response.

Returns

This returns the same elements as the **queryProcessInstances** request, except it does not include the <ap:QueryId> element. See queryProcessTemplates.

Note that this method could return zero items if the number of process instances has decreased since the initial query was made (because of process instances being completed or cancelled since the query).

queryProcessInstancesPreviousPage

This request returns the previous page of process instances — used after calling the **queryProcessInstances** request with the page size set to a value other than 0, then calling the **queryProcessInstancesNext** request to get the next page of process instances.

Note that if you call this request, and there are no process instances to return, an exception is thrown.

Syntax

```
com.tibco.wcc.base.Requests.queryProcessInstancesPrevousPage(requestId,
                                                             queryId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *queryId* - (String) A unique identifier for the **queryProcessInstances** request in which you have requested paging for the returned process instances. The value you must pass in this parameter is returned in the <ap:QueryId> element in the **queryProcessInstances** request response.

Returns

This returns the same elements as the **queryProcessInstances** request, except it does not include the <ap:QueryId> element. See queryProcessTemplates.

Note that this method could return zero items if the number of process instances has decreased since the initial query was made (because of process instances being completed or cancelled since the query).

queryProcessInstancesFirstPage

This request returns the first page of process instances.

This request is used after calling the queryProcessInstances request with the page size set to a value other than 0. You must pass in a query ID that was returned in the response from the **queryProcessInstances** request.

Syntax

```
com.tibco.wcc.base.Requests.queryProcessInstancesFirstPage(requestId,
                                                           queryId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *queryId* - (String) A unique identifier for the **queryProcessInstances** request in which you have requested paging for the returned process instances. The value you must pass in this parameter is returned in the <ap:QueryId> element in the **queryProcessInstances** request response.

Returns

This returns the same elements as the **queryProcessInstances** request, except it does not include the <ap:QueryId> element. See queryProcessTemplates.

queryProcessInstancesLastPage

This request returns the last page of process instances.

This request is used after calling the queryProcessInstances request with the page size set to a value other than 0. You must pass in a query ID that was returned in the response from the **queryProcessInstances** request.

Syntax

```
com.tibco.wcc.base.Requests.queryProcessInstancesLastPage(requestId,
                                                          queryId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *queryId* - (String) A unique identifier for the **queryProcessInstances** request in which you have requested paging for the returned process instances. The value you must pass in this parameter is returned in the <ap:QueryId> element in the **queryProcessInstances** request response.

Returns

This returns the same elements as the **queryProcessInstances** request, except it does not include the <ap:QueryId> element. See queryProcessTemplates.

Note that this method could return zero items if the number of process instances has decreased since the initial query was made (because of process instances being completed or cancelled since the query).

queryDone

This request notifies the server that no more next/previous page requests will be made for a paged list of process instances. This frees up server-side resources for managing the list.

Syntax

```
com.tibco.wcc.base.Requests.queryDone(requestId,
                                      queryId)
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *queryId* - (String) A unique identifier for the **queryProcessInstances** request in which you have requested paging for the returned process instances. The value you must pass in this parameter is returned in the <ap:QueryId> element in the **queryProcessInstances** request response.

Returns

This returns only the <ap:QueryDone> element.

For example:

```
<ap:Requests>
  <ap:QueryDone Id="ApiSample.queryProcessInstancesDone" />
</ap:Requests>
```

suspendProcessInstance

This suspends the specified process instance. This causes the process instance's status to change to SUSPENDED. It also results in all work items associated with the process instance to also become suspended; when suspended, they cannot be opened.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see suspendProcessInstance.

resumeProcessInstance

This resumes (unsuspends) the specified suspended process instance. The specified process instance's status changes from SUSPENDED to ACTIVE. And the associated work items that were suspended when the process instance was suspended are also resumed — their state is changed from SUSPENDED to the state they were in when the process instance was suspended.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see resumeProcessInstance.

cancelProcessInstance

This cancels the specified process instance. This results in all work items associated with that process instance also being cancelled — they are removed from work lists of all users to whom they were offered or allocated.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see cancelProcessInstance.

resumeHaltedProcessInstance

This method resumes a halted process instance so that the instance itself goes into a failed state because of the failed task. This would typically be used if the reason for the failed task cannot be resolved.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see resumeHaltedProcessInstance.

retryProcessInstance

This method causes the previously failed task to be retried so that the halted process instance can progress normally. This method is used if the reason for the failed task has been resolved, for example, if it failed because a database connection had been lost, but has now been restored.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see retryProcessInstance.

ignoreProcessInstance

This method causes the failed task to be skipped in the halted process instance. The process instance continues processing from the point in the process after the failed task.

Note that this is actually a base Application-class method. It is shown in the **wccApiSample** application's `Application.js` only to illustrate how you can use the output of some of the server requests as input to base Application-class methods.

For information about using this method, see ignoreProcessInstance.

Business Service Requests

Server requests are available for working with business services.

listBusinessServiceCategories

This request returns a list of all valid business service categories.

Syntax

```
com.tibco.wcc.base.Requests.listBusinessServiceCategories(requestId,
                                                          channelId,
                                                          includeFormalParams);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *channelId* - (String) Identifies the presentation channel. Note that for WCC applications, always pass the results of **this.getChannelId()** in this parameter.
- *includeFormalParams* - (boolean) (optional: default = false) If true, business services with formal parameters are included in the result.

Returns

Returns an <ap:Category> element for each business service category. Each <ap:Category> element contains the following elements:

- <ap:Name> - The name of the business service category.
- <ap:ChildCategories> - This element can contain additional <ap:Category> elements for nested business service categories.

For example:

```
<ap:Requests>
  <ap:ListBusinessServiceCategories Id="ApiSample.listBusinessServiceCategories">
    <ap:Categories>
      <ap:Category>
        ap:Name>Acme</ap:Name>
        <ap:ChildCategories>
          <ap:Category>
            <ap:Name>Common Fields</ap:Name>
            <ap:ChildCategories />
          </ap:Category>
        </ap:ChildCategories>
      </ap:Category>
              .
              .
              .
    </ap:Categories>
  </ap:ListBusinessServiceCategories>
</ap:Requests>
```

queryBusinessServices

This request returns a list of business services for a specified category.

Also see queryBusinessServiceCategories.

Syntax

```
com.tibco.wcc.base.Requests.queryBusinessServices(requestId,
                                                  category,
                                                  channelId,
                                                  includeFormalParams);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *category* - (String) The name of the category whose business services are to be returned. This can be retrieved from the listBusinessServiceCategories or queryBusinessServiceCategories requests.
- *channelId* - (String) Identifies the presentation channel. Note that for WCC applications, always pass the results of **this.getChannelId()** in this parameter.
- *includeFormalParams* - (boolean) (optional: default = false) If true, business services with formal parameters are included in the result.

Returns

Returns an <ap:BusinessServiceTemplate> element for each business service in the specified category. Each <ap:BusinessServiceTemplate> element contains the following elements:

- <ap:ModuleName> - The path to the XPDL file that defines the "process package" in which the business service is defined.
- <ap:ProcessName> - The name of the process started by the business service.
- <ap:Version> - The version of the process started by the business service.
- <ap:HasFormalParameters> - Indicates whether or not the business service includes formal parameters.

For example:

```
<ap:Requests>
  <ap:QueryBusinessServices Id="ApiSample.queryBusinessServices">
    <ap:BusinessServiceTemplates>
      <ap:BusinessServiceTemplate>
        <ap:ModuleName>/Acme-Contact/Process Packages/ProcessPackage.xpdl</
ap:ModuleName>
        <ap:ProcessName>AskQuestion</ap:ProcessName>
        <ap:Version>1.0.1.201101101429</ap:Version>
        <ap:HasFormalParameters>false</ap:HasFormalParameters>
      </ap:BusinessServiceTemplate>
                      .
                      .
                      .
    </ap:BusinessServiceTemplates>
  </ap:QueryBusinessServices>
</ap:Requests>
```

queryBusinessServiceCategories

This request returns a list of business services for a specified category.

Note that the difference between this request and queryBusinessServices is that this request returns the category name as part of the response so that if multiple requests are supplied in one action, it will be easier to match up categories with services.

Syntax

```
com.tibco.wcc.base.Requests.queryBusinessServiceCategories(requestId,
                                                           category,
                                                           channelId,
                                                           includeFormalParams);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *category* - (String) The name of the category whose business services are to be returned. This can be retrieved from the listBusinessServiceCategories or queryBusinessServiceCategories requests.
- *channelId* - (String) Identifies the presentation channel. Note that for WCC applications, always pass the results of **this.getChannelId()** in this parameter.
- *includeFormalParams* - (boolean) (optional: default = false) If true, business services with formal parameters are included in the result.

Returns

Returns an <ap:BusinessServiceTemplate> element for each business service in the specified category. This response also returns an <ap:BusinessCategory> element, which contains the name of the category, allowing you to match up categories with business service if multiple requests are supplied in one action.

Each <ap:BusinessServiceTemplate> element contains the following elements:

- <ap:ModuleName> - The path to the XPDL file that defines the “process package” in which the business service is defined.
- <ap:ProcessName> - The name of the process started by the business service.
- <ap:Version> - The version of the process started by the business service.
- <ap:HasFormalParameters> - Indicates whether or not the business service includes formal parameters.

For example:

```
<ap:Requests>
  <ap:QueryBusinessServiceCategories Id="ApiSample.queryBusinessServiceCategories">
    <ap:BusinessCategories>
      <ap:BusinessCategory>
        <ap:Name>Acme</ap:Name>
        <ap:BusinessServiceTemplates>
          <ap:BusinessServiceTemplate>
            <ap:ModuleName>/Acme-Contact/Process Packages/ProcessPackage.xpdl</
ap:ModuleName>
            <ap:ProcessName>AskQuestion</ap:ProcessName>
            <ap:Version>1.0.1.201101101429</ap:Version>
            <ap:HasFormalParameters>false</ap:HasFormalParameters>
          </ap:BusinessServiceTemplate>
                      .
                      .
                      .
        </ap:BusinessServiceTemplates>
        <ap:ChildBusinessCategories />
      </ap:BusinessCategory>
    </ap:BusinessCategories>
  </ap:QueryBusinessServiceCategories>
</ap:Requests>
```

Event Collector Requests

Server requests are available for working with the event collector.

eventCollectorGetComponents

This request returns a list of the components that generate events.

Syntax

```
com.tibco.wcc.base.Requests.eventCollectorGetComponents(requestId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

Returns an <ap:Component> element for each component that generate events. Each <ap:Component> element contains the following elements:

- <ap:Id> - The ID of the component.
- <ap:Name> - The name of the component.
- <ap:Version> - The version of the component.
- <ap:Description> - The description of the component.
- <ap:ParentId> - The ID of the parent component. (Components can write to attributes defined in their own component, as well as attributes defined in components higher in the component hierarchical structure.)

For example:

```
<ap:Requests>
  <ap:EventCollectorGetComponents Id="ApiSample.eventCollectorGetComponents">
    <ap:Components Id="ApiSample.eventCollectorGetComponents">
      <ap:Component>
        <ap:Id>1</ap:Id>
        <ap:Name>N2LF</ap:Name>
        <ap:Version>1.1</ap:Version>
        <ap:Description>N2 Logging Framework</ap:Description>
        <ap:ParentId>-1</ap:ParentId>
      </ap:Component>
               .
               .
               .
    </ap:Components>
  </ap:EventCollectorGetComponents>
</ap:Requests>
```

eventCollectorGetAllAttributes

This request returns a list of all event collector attributes.

Also see eventCollectorGetAttributes to return event collector attributes for a specified component.

Syntax

```
com.tibco.wcc.base.Requests.eventCollectorGetAllAttributes(requestId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.

Returns

Returns an <ap:AttributeDefinition> element for each attribute returned. Each <ap:AttributeDefinition> element contains the following elements:

- <ap:Id> - The ID of the attribute.
- <ap:Name> - The name of the attribute.
- <ap:Category> - The category in which the attribute was placed when it was defined.
- <ap:Type> - The type of data stored in the attribute.
- <ap:IsPrimary> - Indicates whether or not the attribute is in the primary Events table. For internal use.
- <ap:ComponentId> - The ID of the component in which the attribute was defined. (Components can write to attributes defined in their own component, as well as attributes defined in components higher in the component hierarchical structure.)
- <ap:IsSortable> - true = the attribute is sortable; false = the attribute is not sortable.
- <ap:IsFilterable> - true = the attribute is filterable; false = the attribute is not filterable.

For example:

```
<ap:Requests>
  <ap:EventCollectorGetAllAttributes Id="ApiSample.eventCollectorGetAllAttributes">
    <ap:AttributeDefinitions Id="ApiSample.eventCollectorGetAllAttributes">
      <ap:AttributeDefinition>
        <ap:Id>1</ap:Id>
        <ap:Name>messageId</ap:Name>
        <ap:Category>message</ap:Category>
        <ap:Type>STRING</ap:Type>
        <ap:IsPrimary>true</ap:IsPrimary>
        <ap:ComponentId>1</ap:ComponentId>
        <ap:IsSortable>true</ap:IsSortable>
        <ap:IsFilterable>true</ap:IsFilterable>
      </ap:AttributeDefinition>
               .
               .
               .
    </ap:AttributeDefinitions>
  </ap:EventCollectorGetAllAttributes>
</ap:Requests>
```

eventCollectorGetAttributes

This request returns a list of all event collector attributes associated with a specified component.

Syntax

```
com.tibco.wcc.base.Requests.eventCollectorGetAttributes(requestId,
                                                         componentId);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *componentId* - (String) Identifies the component whose event collector attributes are to be returned.

Returns

Returns an <ap:AttributeDefinition> element for each component that writes entries to the event collector. Each <ap:AttributeDefinition> element contains the following elements:

- <ap:Id> - The ID of the attribute.
- <ap:Name> - The name of the attribute.
- <ap:Category> - The category in which the attribute was placed when it was defined.
- <ap:Type> - The type of data stored in the attribute.
- <ap:IsPrimary> - Indicates whether or not the attribute is in the primary Events table. For internal use.
- <ap:ComponentId> - The ID of the component in which the attribute was defined. (Components can write to attributes defined in their own component, as well as attributes defined in components higher in the component hierarchical structure.)
- <ap:IsSortable> - true = the attribute is sortable; false = the attribute is not sortable.
- <ap:IsFilterable> - true = the attribute is filterable; false = the attribute is not filterable.

For example:

```
<ap:Requests>
  <ap:EventCollectorGetAttributes Id="ApiSample.eventCollectorGetAttributes">
    <ap:AttributeDefinitions Id="ApiSample.eventCollectorGetAttributes">
      <ap:AttributeDefinition>
        <ap:Id>31</ap:Id>
        <ap:Name>resourceName</ap:Name>
        <ap:Category />
        <ap:Type>STRING</ap:Type>
        <ap:IsPrimary>true</ap:IsPrimary>
        <ap:ComponentId>2</ap:ComponentId>
        <ap:IsSortable>true</ap:IsSortable>
        <ap:IsFilterable>true</ap:IsFilterable>
      </ap:AttributeDefinition>
               .
               .
               .
    </ap:AttributeDefinitions>
  </ap:EventCollectorGetAttributes>
</ap:Requests>
```

eventCollectorQuery

This request returns a list of events that match a specified filter expression.

Syntax

```
com.tibco.wcc.base.Requests.eventCollectorQuery(requestId,
                                                bCorrelate,
                                                txtFilter,
                                                arrSorts,
                                                bReturnAttributes,
                                                numStartIndex,
                                                numReturnCount);
```

Parameters

- *requestId* - (String) Uniquely identifies the request. For more information, see Submitting Server Requests.
- *bCorrelate* - (boolean) true = return all events that satisfy the filter expression, *plus* the events that are correlated to those events. false = return all events that satisfy the filter expression, but not the correlated events.

- *txtFilter* - (String) - The filter expression to apply to the events. For information about filter syntax, see the "Defining Query Filter Strings" topic in the developer's guide for your BPM runtime environment.

  For example:

  ```
  (  componentId = 'BRM'  OR  componentId = 'DE'  )  AND  (  priority = 'MEDIUM'  )
  ```

- arrSort - (Array[Object]) The attributes used to sort the events returned. The events are returned in order according to the value in the specified attributes.

  ```
  [
    {
       Attribute: Attribute,
       Type: Type,
    }
  ]
  ```

  where:

- *Attribute* - (String) The name of the attribute used for sorting the events.
- *Type* - (String) (optional) The type of sort: ASCENDING or DESCENDING.
- *bReturnAttributes* - (boolean) Specifies whether or not to return attribute values — can be used to just get the count. Also note that if you choose not to return attributes, elements representing the attributes are still returned in the response, but they do not contain any values.
- *numStartIndex*- (Number) The starting index (zero-based) number for the returned events.
- *numReturnCount* - (Number) The number of events to return per request.

Returns

Returns the following elements:

- <ap:StartPosition> - The starting index for the event request.
- <ap:EndPosition> - The ending index for the event request.
- <ap:TotalItems> - The total number of events returned (the number that satisfied the filter expression).
- <ap:Events> - Contains an <ap:Event> element for each event that is returned. Each <ap: Event> element contains the following elements:

  – <ap:Attributes> - Contains an <ap:Attribute> element for each attribute that is returned for the event. Each <ap:Attribute> element contains the following elements:

  – <ap:AttributeId> - A unique identifier for the attribute.

  – <ap:Value> - The value assigned to the attribute.

For example:

```
<ap:Requests>
  <ap:EventCollectorQuery Id="ApiSample.eventCollectorQuery">
    <ap:StartPosition>0</ap:StartPosition>
    <ap:EndPosition>20</ap:EndPosition>
    <ap:TotalItems>251</ap:TotalItems>
    <ap:Events>
      <ap:Event>
        <ap:Attributes>
          <ap:Attribute>
            <ap:AttributeId>27</ap:AttributeId>
            <ap:Value>10.97.8.142</ap:Value>
          </ap:Attribute>
          <ap:Attribute>
            <ap:AttributeId>26</ap:AttributeId>
            <ap:Value>BPMNode</ap:Value>
          </ap:Attribute>
          <ap:Attribute>
```

```
            <ap:AttributeId>24</ap:AttributeId>
            <ap:Value>SECURITY</ap:Value>
          </ap:Attribute>
               .
               .
               .
        </ap:Attributes>
      </ap:Event>
    </ap:Events>
  </ap:EventCollectorQuery>
</ap:Requests>
```

Callout Interface Methods

The callout interface methods allow you to *callout*, or *inject*, custom specifications in WCC applications, including the Workspace application.

The callout interface methods can be used to do the following:

- Define custom filters, sorts, filter attributes, sort attributes, and display columns on various lists.
- Restrict the list of processes that are available to a user when creating a process view or starting a process instance.
- Restrict the list of business services or business service categories that are available to a user when starting a business service.
- Specify characteristics of lists displayed in the application, for example, font color, alignment, etc. (This is accomplished using the modifyMatrix method.)

These methods can be used in conjunction with user access profile settings to control filter, sort, and attribute display for various entities in the organizational model. The content and characteristics of lists may be managed by invoking these methods when the list initially loads, or when the user modifies filter, sort, or column definitions.

The following summarizes some example use cases for the methods in the callout interface. Note that this is not an exhaustive list; the callout methods can be used for additional uses case beyond those listed here.

- When the application initially starts (that is, the very first time), apply a default filter or sort to a list that varies depending on the organizational group to which the logged-in user belongs. However, allow the user to modify the default filter and sorting settings, which will apply in future sessions.

  To implement this use case, use the following methods:

  – overrideFilterOnLoad

  – overrideSortOnLoad
- Prohibit all members of a particular organizational position from filtering, sorting, or displaying a certain set of attributes.

  To implement this use case, use the following methods:

  – overrideFilterAttributes

  – overrideSortAttributes
- Apply an undisclosed filter to a list to restrict access to sensitive data (although users can still apply additional filters).

  To implement this use case, use the following method:

  – overrideFilterOnRefresh
- Display, for each list type, a default set of columns, which varies depending on the organizational group to which the logged-in user belongs. Users may modify the default columns, but regardless of

the columns chosen by the user, a certain attribute must always be displayed in the first column and the list must be sorted on this attribute at all times.

To implement this use case, use the following methods:

  - overrideColumnsOnLoad (for the initial default columns)
  - overrideColumnsOnRefresh (to force the first column and sort on attribute)

- Apply a particular filter and sort order to each list type, which may *not* be modified by the user, that is, the user is denied access to the tools for filtering and sorting.

  To implement this use case, use the following methods:

  - overrideFilterOnLoad
  - overrideSortOnLoad
  - "Filter" and "Sort" removed from `userAccess.xml` to deny access to filter and sort tools.

- Display a label (for example, "SSN") for a particular work item attribute in the user's Inbox work view, but display a different label (for example, "Customer ID") for the same attribute in other work views.

  To implement this use case, use the following method:

  - overrideColumnsOnRefresh

- Display values in a particular list column in a larger, bolder font with a different color than other columns in the list.

  To implement this use case, use the following method:

  - modifyMatrix

- Localize or customize the names of work views, process views, and event views, as well as specify the order of the views in the views list.

  To implement this use case, use the following method:

  - modifyViewListData

In order to implement the callout interface, a callout controller class is provided in the WCC base application (`com.tibco.wcc.base`) and a customized callout handler class (**com.tibco.wcc.*appName*.CalloutHandler**) is loaded by the WCC application.

The callout handler contains the business-specific logic to control filters, sorts and attributes. This logic is contained in "override" callout methods, which are registered with the callout controller when the handler class is loaded. The controller maintains a list of the available callout methods and invokes the appropriate method as application events occur (such as creating a list or applying a filter).

Some of the callout interface methods require information that is not readily available, such as user details, or the XML that represents the graphical representation of a filter expression. Some "helper" methods are provided that can be used to obtain this type of information. These are described in Callout Utilities.

Some of the callout methods can be used to filter and sort the attributes that are shown on work item, process instance, and event lists. However, only some attributes are filterable and sortable. For more information, see Workspace Attributes.

Sample CalloutHandler

A sample callout handler (`CalloutHandler.js`) is provided that illustrates the use of the callout interface methods.

This is the same callout handler you will use to implement custom callouts; part of the configuration process for custom callouts is to copy the sample callout handler to the proper location, then modify it to fit your needs (see Callout Handler Configuration).

An example TIBCO Business Studio project containing an organization model and business process is also provided with the sample callout handler (in the same `\Samples\Callouts\` directory). The sample implementations of the callout methods in `CalloutHandler.js` make use of the specific organizational entities and processes defined in the example TIBCO Business Studio project.

To see the sample implementations in action, you will need to deploy the example project to a running node. Or you can modify the method calls in the sample callout handler to include specifics about your own deployed projects (that is, your own organizational entities and business processes).

Callout Handler Configuration

You must configure your system to use the callout interface.

Procedure

1. Copy the following directory:

   ```
   StudioHome\wcc\version\Samples\Callouts\
   ```

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.

   The path shown above is in the context of a custom WCC application in a TIBCO Business Studio development environment. If you want to implement callouts in the Workspace application that is pre-deployed to a BPM runtime machine, substitute the `StudioHome\wcc\version\` portion of the path shown above with the following path in all steps in the Configuration section:

   ```
   %config_home%\tibcohost\Admin-environmentName-adminServer\data_3.2.x\host
   \plugins\com.tibco.n2.rtc.ws_Version\resources\
   ```

   where:

   - *%config_home%* is the root directory for BPM runtime files. This can be specified during the installation. On Windows systems, this defaults to: `C:\ProgramData\amx-bpm\tibco\data`; on UNIX systems, this defaults to: `/opt/tibco/data`
   - *environmentName* - The name of the BPM environment. This can be specified during the installation.
   - *adminServer* - The name of the TIBCO Administrator server. This can be specified during the installation.
   - *Version* is the specific version of the Workspace software installed.

2. Paste the `...\Samples\Callouts\` directory that you copied in step 1 into the following directory:

   ```
   StudioHome\wcc\version\JSXAPPS\
   ```

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio

3. Optionally rename the `CalloutHandler.js` file something more specific if that's what its purpose is, for example `ColumnsCalloutHandler.js`.

   Note that you can have multiple callout handlers, each one performing different types of callout functions, for example, `ColumnsCalloutHandler.js`, `FiltersCalloutHandler.js`, etc.

   If you have multiple callout handlers, they all need to be specified in the application's `config.xml` file; see step 5.

4. Open the callout handler file in the `...\JSXAPPS\Callouts\` directory and modify it for the callouts you want to implement, as follows:

   a) Locate the `registerHandler` function and ensure that the callout methods you are implementing are uncommented. Note that some of the methods are uncommented by default. Comment those out if you are not implementing those methods.

   b) Locate the method(s) you want to implement and modify them as follows:

      - All of the callout methods have two parameters: *oValue* and *oContext* (the lone exception is the modifyMatrix method, which has *oMatrix* and *oContext* parameters):

        oValue - An object with properties that are determined by the type of callout. The properties of this object are modified as required by the callout and returned as output of the method.

        oContext - This object provides information about the list being modified, as well as information about the logged-in user. This includes properties such as `oContext.userName` (name of the logged-in user) and `oContext.userGuid` (unique identifier of the logged-in user). It may include additional properties, depending on the type of callout.

      - Each callout method has a SAMPLE CHANGE example that illustrates making a modification to a filter, sort, column, etc., for a particular list type.

      - Each of the callout methods contains `if/else` statements that provide a location to place code that modifies the *oValue* object for each of the list types for which the method applies.

      - You can easily enable or disable the method for each list type by either commenting (to enable) or uncommenting (to disable) the `oValue = null;` line in the appropriate `if/else` segment for the desired list type.

   c) Save and close the callout handler file.

5. Open and update your WCC application's `config.xml` file as described below.(1) This file is located as follows:

   ```
   StudioHome\wcc\version\JSXAPPS\WCCProjectName\config.xml
   ```

   where:

   - *`StudioHome`* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.
   - *WCCProjectName* is the name of the General Interface Builder project that contains your WCC application. If you are working with the Workspace application, this is "workspace".

   a) Locate the <record jsxid="customCallout"> element and specify a <Class> element for each callout handler to implement. This must include the fully qualified name of the callout handler. For example, if your WCC application is named "accounts", it must appear as follows:

   ```
   <record jsxid="customCallout" type="Workspace">
      <Classes>
         <Class class="com.tibco.wcc.accounts.CalloutHandler" />
      </Classes>
   </record>
   ```

[6] If you are configuring custom callout handlers for a deployed, running application, you can use the Configuration Administrator to update the application's configuration file, rather than opening and updating the file (`config.xml`) directly. For information, see Using Configuration Administrator to Configure Callout Handlers.

Note that the `<Classes>` element can contain multiple `<Class>` elements if you have created multiple callout handlers to handle different types of callouts (for example, `ColumnsCalloutHandler.js`, `FiltersCalloutHandler.js`, etc.).

b) Locate and uncomment the example mapping record for the callout handler:

```
<!--<record jsxid="com.tibco.wcc.workspace.CalloutHandler" type="map">
<record jsxid="id" type="string">CalloutHandler</record>
<record jsxid="type" type="string">script</record>
<record jsxid="owner" type="string">application</record>
<record jsxid="onLoad" type="boolean">true</record>
<record jsxid="required" type="boolean">true</record>
<record jsxid="src"        type="string">JSXAPPS/Callouts/CalloutHandler.js</
record>
</record> -->
```

c) In the mapping record that you have uncommented, ensure that the `id` and `src` records match the name and location of your callout handler.

d) If needed, modify the value of the `jsxid` attribute in the top-level record (the one in which `type="map"`). By default, the value is:

```
jsxid="com.tibce.wcc.workspace.CalloutHandler"
```

Note that the value of this attribute *must* match the class name for the callout handler in the `customCallout` record (see step 5a above).

Therefore, if you are using the Workspace application, the default value of the `jsxid` attribute in the top-level record may be fine (assuming your callout handler is called “CalloutHandler”).

If, however, you are configuring a custom WCC application, you will need to modify the value of the `jsxid` attribute to include the name of your WCC application. For example, if the name of your WCC application is “accounts”, the value of the `jsxid` attribute may need to be changed to:

jsxid="com.tibce.wcc.accounts.CalloutHandler"

Again, this assumes the name of your callout handler is “CalloutHandler”. If it is something like “ColumnsCalloutHandler”, include that in the value.

e) If you are configuring multiple callout handlers (for example, FilterCalloutHandler and SortCalloutHandler), copy and paste the mapping record that you uncomment in step 5b above, and modify its contents according to the instructions in steps 5c and 5d.

For example, if you have two callout handlers, FilterCalloutHandler and SortCalloutHandler, you need two mapping records (as well as two `class` elements in the `customCallout` record; see step 5a above):

```
<record jsxid="com.tibco.wcc.workspace.FilterCalloutHandler" type="map">
  <record jsxid="id" type="string">FilterCalloutHandler</record>
  <record jsxid="type" type="string">script</record>
  <record jsxid="owner" type="string">application</record>
  <record jsxid="onLoad" type="boolean">true</record>
  <record jsxid="required" type="boolean">true</record>
  <record jsxid="src" type="string">JSXAPPS/Callouts/FilterCalloutHandler.js</
record>
</record>

<record jsxid="com.tibco.wcc.workspace.SortCalloutHandler" type="map">
  <record jsxid="id" type="string">SortCalloutHandler</record>
  <record jsxid="type" type="string">script</record>
  <record jsxid="owner" type="string">application</record>
  <record jsxid="onLoad" type="boolean">true</record>
  <record jsxid="required" type="boolean">true</record>
  <record jsxid="src" type="string">JSXAPPS/Callouts/SortCalloutHandler.js</
record>
</record>
```

f) Save and close the `config.xml`

If you configure multiple callout handlers, and they implement the same callout method, only the method in the last handler that is loaded is executed.

6. Optionally modify user access profiles that are to be used on conjunction with the callouts. For example, if your custom handler is setting the default columns on the work item list, you may want

to deny access to "SelectColumns" for the work item list in your application’s `userAccess.xml` file. For information about setting user access profiles, see the *TIBCO Workspace Customization and Configuration* guide.

7. If you are implementing callouts in a custom WCC application, you will need to deploy the application to a node after you are finished customizing it. For information, see the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Customization and Configuration* guide.

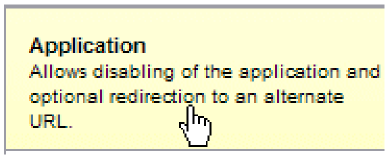
Using Configuration Administrator to Configure Callout Handlers

If you are configuring a custom callout handler for a deployed, running application, you can use the Configuration Administrator to update the application’s configuration file, rather than opening and updating the file (`config.xml`) directly.

The instructions here take the place of the instructions in step 5 of the Callout Handler Configuration topic.

Procedure

1. Access the Configuration Administrator from Workspace by clicking the **Admin** button in the Workspace toolbar, then selecting **Configuration**.
2. In the Configuration Administrator, click on `config.xml` in the left pane.
3. In the **Graphical Editor** tab, click in the **Application** section:



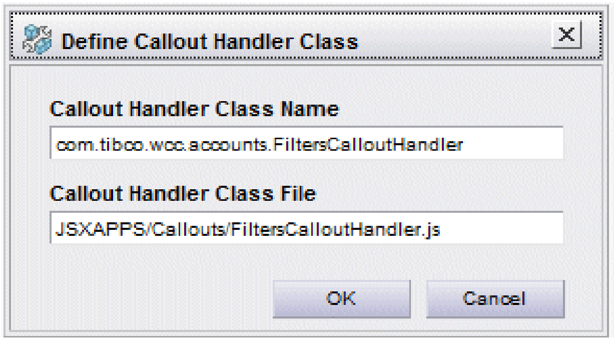
Application
Allows disabling of the application and optional redirection to an alternate URL.

4. Select the **Enable Callout Interfaces** box.
5. Click the **Add** button.
6. In the **Callout Handler Class Name** field on the Define Callout Handler Class dialog, enter the fully qualified name of the callout handler class. This must be in the form:
   ```
   com.tibco.wcc.appName.calloutClassName
   ```
   where:

   - *appName* is the name of the WCC application. This is “workspace” if you are using the Workspace application, or the name given to your application if you are using a custom WCC application.
   - *calloutClassName* is the name of the callout handler. The name of the default callout handler is “CalloutHandler”, but if you are using a custom callout handler, this could be something like “FiltersCalloutHandler”.
7. In the **Callout Handler Class File** field on the Define Callout Handler Class dialog, enter the path to callout handler file, starting from the `JSXAPPS` directory. For example:
   ```
   JSXAPPS/Callouts/calloutFileName
   ```
   where:

   - *calloutFileName* is the name of the callout handler file.

     The following shows an example Define Callout Handler Class dialog:



Callout Handler Class Name: com.tibco.wcc.accounts.FiltersCalloutHandler

Callout Handler Class File: JSXAPPS/Callouts/FiltersCalloutHandler.js

8. Click **OK** to close the Define Callout Handler Class dialog.
9. Repeat step 5 - step 8 if you have multiple callout handlers.
10. Click **OK** to close the Configuration Administrator.
11. Return to step 6 of the Callout Handler Configuration topic to complete the callout handler configuration.

Callout Methods

The following provides a brief summary of the callout interface methods:

- overrideBaseFilter - Sets the base filter for a work view, process view, or event view. It is called when a view definition is created or edited, and whenever the list contents are refreshed.
- overrideFilterOnLoad - This method is called once while initializing the list and provides a means for modifying the refinement filter initially applied to a work item, process instance, or event list.
- overrideFilterOnRefresh - This method is called whenever a list is initially opened, each time the list is refreshed, and each time the user specifies a filter in the Filter dialog. It provides a means for always appending a filter expression to any filters that may be defined by the user.

  Filter changes applied by this method do NOT appear in the Filter dialog and are not visible to the user.
- overrideFilterAttributes - Defines the list of attributes available for filtering views of work items, process instances, and events. This is called once while initializing the list of filter attributes in the Filter dialog.
- overrideSortOnLoad - This method is called once while initializing the list and provides a means for modifying the sort parameters initially applied to a work item, process instance, or event list.
- overrideSortOnRefresh - This method is called whenever a list is initially opened, each time the list is refreshed and each time the user specifies a sort in the Sort dialog. It provides a means for always appending a sort specification to any sort that may be defined by the user.

  Sort changes applied by this method do NOT appear in the Sort dialog and are not visible to the user.
- overrideSortAttributes - Defines the list of attributes available for sorting views of work items, process instances, and events. This is called once while initializing the list of sort attributes in the Sort dialog.
- overrideColumnsOnLoad - Defines the columns (attributes) that display, by default, for views of work items, process instances, and events. This is called once while initializing the list.
- overrideColumnsOnRefresh - This method is called whenever a list is initially opened, each time the list is refreshed and each time the user specifies columns in the Column Selector dialog. It provides a means for always appending columns or removing columns defined by the user.

- overrideColumnAttributes - Defines the list of attributes available for display as columns in views of work items, process instances, and events. This is called once while initializing the list of attributes in the Column Selector dialog.
- overrideProcesses - Defines the list of processes that are available to a user when creating a process view or starting a process instance. This is called once while initializing the list of available processes.
- overrideBusinessServices - Defines the list of business services that are available to a user when starting a business service. This is called once while initializing the list of available services.
- overrideBusinessServiceCategories - Defines the list of business service categories that are available to a user when starting a business service. This is called once while initializing the list of available business services.
- modifyMatrix - This method provides a means for modifying the characteristics of lists displayed in the application. A reference to a TIBCO General Interface `jsx3.gui.Matrix` component is provided as input and may be modified by the method. This is called once when the list is first constructed.
- modifyViewListData - This method provides a means of modifying display values for work items in work view, process view, and event view lists. The XML that populates the list is passed to this method, providing a means to manipulate list data prior to it being displayed. Text values of view names and descriptions may be changed, allowing for localization or customization, plus items in the list may be reordered.

Callout methods that affect views do *not* have an affect on the “master” copy of system views; they will affect only the “recipient” copy of a system view. For more information about system views, see the *Using Views* chapter in the *TIBCO Workspace User’s Guide.*

overrideBaseFilter

This method provides a means for modifying the base filter for a work view, process view, or event view. It is called when a view definition is created or edited, and whenever the list contents are refreshed.

A context property, passed into this method, indicates the type of action that caused the callout method to be executed. This property may be utilized to set default base filters that are visible and may be modified in the view wizard, or to create base filters that are not visible or accessible to the user and are applied every time the list is refreshed.

Syntax

```
classProto.overrideBaseFilter = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the following properties, defining the base filter:
  - *oValue.filter* - (string) The base filter expression to initially apply to the view. For information about viewing filter expressions on the Filter dialog in a WCC application, see Callout Utilities. For additional details about filter expressions, see the “Sorting List" and "Filtering Lists” topics in the *TIBCO ActiveMatrix BPM Developer’s Guide*.
  - *oValue.filterXML* - (string) The XML that is used to graphically depict the filter expression in the Filter dialog. This XML string is obtained using the **Export Filter XML** tool on the Filter dialog in a WCC application (see Callout Utilities).
  - *oValue.userChanged* - (boolean read-only) A flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value is null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.

- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
  - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - *oContext.viewName* - (string read-only) The name of the view that created the list.
  - *oContext.viewDescription* - (string read-only) A description of the view.
  - *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.
  - *oContext.viewAction* - (string read-only) One of the following, depending on what action has caused the callout to be invoked:

    "onCreate" - The view wizard has been opened for the purpose of creating a new view. Use the callout method to specify the default filter for the view. This filter is displayed on the Filter dialog of the wizard and may be modified by the user.

    "onEdit" - The view wizard has been opened for the purpose of editing an existing view. Use the callout method to modify the existing base filter defined for the view. This filter is displayed on the Filter dialog of the wizard and may be modified by the user.

    "onRefresh" - The list is initially loading or has been refreshed for any reason. Use the callout method to apply a base filter that is not visible and cannot be modified by the user in the view wizard.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideFilterOnLoad

This method is called once while initializing the list and provides a means for modifying the refinement filter initially applied to a work item, process instance, or event list. The filter can be modified by the user using the Filter tool on the list.

To prevent users from modifying the refinement filter, remove access to "Filter" in `userAccess.xml`.

Syntax

```
classProto.overrideFilterOnLoad = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the following properties, defining the base filter:
  - *oValue.filter* - (string) The filter expression to initially apply to the view. For information about viewing filter expressions on the Filter dialog in a WCC application, see Callout Utilities. For additional details about filter expressions, see the “SortingLists" and "Filtering Lists” topics in the *TIBCO ActiveMatrix BPM Developer’s Guide*.
  - *oValue.filterXML* - (string) The XML that is used to graphically depict the filter expression in the Filter dialog. This XML string is obtained using the **Export Filter XML** tool on the Filter dialog in a WCC application (see Callout Utilities).

- *oValue.userChanged* - (boolean read-only) A flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value is null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.

- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
  - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - *oContext.viewName* - (string read-only) The name of the view that created the list.
  - *oContext.viewDescription* - (string read-only) A description of the view.
  - *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideFilterOnRefresh

This method is called whenever a list is initially opened, each time the list is refreshed, and each time the user specifies a filter in the Filter dialog. It provides a means for always appending a filter expression to any filters that may be defined by the user.

Filter changes applied by this method do NOT appear in the Filter dialog and are not visible to the user. To prevent users from defining a filter, remove access to "Filter" in `userAccess.xml`.

Syntax

```
classProto.overrideFilterOnRefresh = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the following properties, defining the base filter:
  - *oValue.filter* - (string) The filter expression to apply to the view. For information about viewing filter expressions on the Filter dialog in a WCC application, see Callout Utilities. For additional details about filter expressions, see the “Sorting Lists" and "Filtering Lists” topic in the *TIBCO ActiveMatrix BPM Developer’s Guide*.
  - *oValue.userChanged* - (boolean read-only) A flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value is null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.

- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.

- *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
- *oContext.viewName* - (string read-only) The name of the view that created the list.
- *oContext.viewDescription* - (string read-only) A description of the view.
- *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideFilterAttributes

This method defines the list of attributes available for filtering views of work items, process instances, and events. This method is called once while initializing the list of filter attributes in the Filter dialog.

A list of all filterable attributes is provided as input to this method. The method may be customized to output an altered list by modifying properties of attributes or removing specific attributes, making them unavailable for filtering.

For a complete list of available attributes, see: Workspace Attributes.

Syntax

```
classProto.overrideFilterAttributes = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the attributes available for defining list filters:
    - *oValue.availableAttributes[]* - (Array of objects) Each object has the following properties, which identify the attribute in the Filter dialog. Property values may be modified (except for "id" and "type") to change the attribute's appearance.
    - *id* - (string read-only) Server identifier for the attribute.
    - *type* - (string read-only) Type of data stored in the attribute. The possible values are:
        - boolean
        - date
        - dateTime
        - double
        - int
        - integer
        - short
        - string
        - time
    - *label* - (string) Text identifying the attribute in the list of available filter attributes. The label is initially obtained from the application language resource file: `JSXAPPS\base\locale\locale.xml` for English, `locale.ll_CC.xml` for other supported languages (where "ll" is the language code and "CC" is the country code). The callout may modify the text value of the label.

However, if multiple languages are supported, a call to the `this.app.getLocaleKey()` method should be made to determine the language currently selected for the application, and set the label to the appropriately translated value.

- *image* - (string) URL (relative to application root) for the icon associated with the attribute.
- *operator* - (string) Default selection in the operator field on the Filter dialog when a filter attribute is selected. The possible values are:
  - Equal
  - NotEqual
  - GreaterThan
  - LessThan
  - GreaterOrEqualTo
  - LessOrEqualTo
- *value* - (string) Default setting in the value field on the Filter dialog when a filter attribute is selected. The value must match the data type specified in the value of the "type" property.

- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
  - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:

    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)

    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)

    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - *oContext.viewName* - (string read-only) The name of the view that created the list.
  - *oContext.viewDescription* - (string read-only) A description of the view.
  - *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideSortOnLoad

This method is called once while initializing the list and provides a means for modifying the sort parameters initially applied to a work item, process instance, or event list.

The sort specification appears in the Sort dialog and may be modified by the user. To prevent users from modifying the sort, remove access to "Sort" in `userAccess.xml`.

Syntax

```
classProto.overrideSortOnLoad = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing properties defining the sort specification:
  - *oValue.sort[]* - (Array of objects) Each object has the following properties that identify a sort attribute:

        - *id* (string) Server identifier for the attribute.
        - *isAscending* (boolean) true or false, indicating whether the sort order should be ascending (true) or descending (false).
    - *oValue.userChanged* - (boolean) A property containing a flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value will be null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
    - *oContext.userName* - (string read-only) The name of the logged-in user.
    - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
    - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
        - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
        - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
        - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
    - *oContext.viewName* - (string read-only) The name of the view that created the list.
- *oContext.viewDescription* - (string read-only) A description of the view.
- *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideSortOnRefresh

This method is called whenever a list is initially opened, each time the list is refreshed and each time the user specifies a sort in the Sort dialog. It provides a means for always appending a sort specification to any sort that may be defined by the user.

Sort changes applied by this method do NOT appear in the Sort dialog and are not visible to the user. To prevent users from defining a sort, remove access to "Sort" in `userAccess.xml`.

Syntax

```
classProto.overrideSortOnRefresh = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing properties defining the sort specification:
    - *oValue.sort[]* - (Array of objects) Each object has the following properties that identify a sort attribute:
        - *id* - (string) Server identifier for the attribute.
        - *isAscending* - (boolean) true or false, indicating whether the sort order should be ascending (true) or descending (false).
    - *oValue.userChanged* - (boolean) A flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value will be null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.

- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
  - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - *oContext.viewName* - (string read-only) The name of the view that created the list.
- *oContext.viewDescription* - (string read-only) A description of the view.
- *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideSortAttributes

This method defines the list of attributes available for sorting views of work items, process instances, and events. This method is called once while initializing the list of sort attributes in the Sort dialog.

A list of all sortable attributes is provided as input to this method. The method may be customized to output an altered list by modifying properties of attributes or removing specific attributes, making them unavailable for sorting.

For a complete list of available attributes, see Workspace Attributes.

Syntax

```
classProto.overrideSortAttributes = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the attributes available in the Sort dialog:
  - *oValue.availableAttributes[]* - (Array of objects) Each object has the following properties that identify an available sort attribute.
  - *id* - (string read-only) Server identifier for the attribute.
  - *label* - (string) Text identifying the attribute in the list of available sort attributes. The label is initially obtained from the application language resource file: `JSXAPPS\base\locale\locale.xml` for English, `locale.ll_CC.xml` for other supported languages (where "ll" is the language code and "CC" is the country code). The callout may modify the text value of the label. However, if multiple languages are supported, a call to the `this.app.getLocaleKey()` method should be made to determine the language currently selected for the application, and set the label to the appropriately translated value.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.

- *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
- *oContext.viewName* - (string read-only) The name of the view that created the list.
- *oContext.viewDescription* - (string read-only) A description of the view.
- *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideColumnsOnLoad

This method defines the columns (attributes) that display, by default, for views of work items, process instances, and events. This function is called once while initializing the list.

A list of system-defined default columns is provided as input to this method. The method may be customized to output an altered list of default columns by modifying properties of columns, removing specific columns, or adding new default columns. To prevent users from modifying the display columns, remove access to "SelectColumns" in `userAccess.xml`.

For a complete list of available attributes, see Workspace Attributes.

Syntax

```
classProto.overrideColumnsOnLoad = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object that specifies the columns to display with the following properties:
    - *oValue.columns[]* - (Array of objects) Each object has the following properties that identify a column:
    - *id* - (string) Identifier for the attribute displayed in the column.
    - *header* - (string) Text identifying the attribute that displays in the column header.
    - *width* - (string) The width of the column (in pixels); a default will be used if this is null.
    - *tooltip* - (string) Text that displays when the cursor hovers over the column header.
    - *oValue.userChanged* - (boolean) A flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value will be null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
    - *oContext.userName* - (string read-only) The name of the logged-in user.
    - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
    - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
        - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)

  - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
  - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
- *oContext.viewName* - (string read-only) The name of the view that created the list.
- *oContext.viewDescription* - (string read-only) A description of the view.
- *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideColumnsOnRefresh

This method is called whenever a list is initially opened, each time the list is refreshed, and each time the user specifies columns in the Column Selector dialog. It provides a means for always appending columns or removing columns defined by the user.

To prevent users from modifying the columns, remove access to "SelectColumns" in `userAccess.xml`.

Syntax

```
classProto.overrideColumnsOnRefresh = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object that specifies the columns to display with the following properties:
  - *oValue.columns[]* - (Array of objects) Each object has the following properties that identify a column:
    - *id* - (string) Identifier for the attribute displayed in the column.
    - *header* - (string) Text identifying the attribute that displays in the column header.
    - *width* - (string) The width of the column (in pixels); a default will be used if this is null.
    - *tooltip* - (string) Text that displays when the cursor hovers over the column header.
  - *oValue.userChanged* - (boolean) A flag indicating whether or not the user has ever modified the value, facilitating logic related to default values. This value will be null if it is unknown whether the user has changed the value because the view was created prior to implementation of the callout interface.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier for the logged-in user.
  - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - *oContext.viewName* - (string read-only) The name of the view that created the list.
  - *oContext.viewDescription* - (string read-only) A description of the view.
  - *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideColumnAttributes

This method defines the list of attributes available for display as columns in views of work items, process instances, and events. This function is called once while initializing the list of attributes in the Column Selector dialog.

A list of all available attributes is provided as input to this method. The method may be customized to output an altered list by modifying properties of attributes or removing specific attributes, making them unavailable for display as columns.

For a complete list of available attributes, see Workspace Attributes.

Syntax

```
classProto.overrideColumnAttributes = function(oValue, oContext)
```

Parameters

- *oValue* (Object) An object containing the attributes available in the Column Selector dialog:
  - *oValue.availableAttributes[]* - (Array of objects) Each object has the following properties that identify an available column attribute.
  - *id* - (string read-only) Server identifier for the attribute.
  - *label* - (string) Text identifying the attribute in the list of available columns. The label is initially obtained from the application language resource file: `JSXAPPS\base\locale\locale.xml` for English, `locale.ll_CC.xml` for other supported languages (where "ll" is the language code and "CC" is the country code). The callout may modify the text value of the label. However, if multiple languages are supported, a call to the `this.app.getLocaleKey()` method should be made to determine the language currently selected for the application, and set the label to the appropriately translated value.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier of the logged-in user.
  - *oContext.viewType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - *oContext.viewName* - (string read-only) The name of the view that created the list.
  - *oContext.viewDescription* - (string read-only) A description of the view.
  - *oContext.isSystemView* - (boolean read-only) True if the view is a system view; False if the view is a user-defined view.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideProcesses

This method defines the list of processes that are available to a user when creating a process view or starting a process instance. This function is called once while initializing the list of available processes.

When creating a new process view, this method can be used to restrict the list of available processes when “Include selected processes” is selected in the view creation wizard. If “Include all processes” is selected, the list is not restricted by this method; all processes are listed.

Note that even though a user is restricted by this method to certain processes when creating a process view, the user may still have access to instances of the restricted processes if a system view had been created for the user by another user.

A list of all available processes is provided as input to this method. The method may be customized to output an altered list by removing specific processes, making them unavailable for selection.

Syntax

```
classProto.overrideProcesses = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the following property, which specifies the processes available in the New Process View dialog or Start Process Instance dialog:
  - *oValue.availableProcesses[]* - (Array of objects) Each object has the following properties that identify an available process:
  - *processName* - (string read-only) Name of the process.
  - *version* - (string read-only) Version of the process.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier of the logged-in user.
  - *oContext.listType* - (string read-only) Either "ProcessView" for a list of processes available when creating a process view, or "StartInstance" for a list of processes available for starting an instance.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideBusinessServices

This method defines the list of business services that are available to a user when starting a business service. This function is called once while initializing the list of available business services.

A list of all available business services is provided as input to this method. The method may be customized to output an altered list by removing specific business services, making them unavailable for selection.

Syntax

```
classProto.overrideBusinessServices = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the available business services:
  - *oValue.availableServices[]* - (Array of objects) Each object has the following properties that identify an available business service:
  - *processName* - (string read-only) Name of the business service.
  - *version* - (string read-only) Version of the business service.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier of the logged-in user.

Returns

The modified `oValue` object, or null if no changes are to be made.

overrideBusinessServiceCategories

This method defines the list of business service categories that are available to a user when starting a business service. This function is called once while initializing the list of available business services.

A list of all available categories is provided as input to this method. The method may be customized to output an altered list by removing specific categories, making them, and all the business services within them, unavailable for selection.

Syntax

```
classProto.overrideBusinessServiceCategories = function(oValue, oContext)
```

Parameters

- *oValue* - (Object) An object containing the available business service categories:
  - *oValue. availableCategories []* - (Array of objects) Each object has the following properties that identify an available business service category.
  - *categoryName* - (string read-only) Name of the business service category.
  - *categoryDescription* - (string) Description of the business service category.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier of the logged-in user.

Returns

The modified `oValue` object, or null if no changes are to be made.

modifyMatrix

This method provides a means for modifying the characteristics of lists displayed in the application. A reference to a TIBCO General Interface `jsx3.gui.Matrix` component is provided as input and may be modified by the method. This method is called once when the list is first constructed.

Properties of the matrix component may be modified, as well as properties of columns (TIBCO General Interface `jsx3.gui.Matrix.Column` components), which are children of the matrix. Some properties that may be changed include the following:

- jsx3.gui.Matrix
  - `setHeaderHeight()` - sets the height of the column header
  - `setRowHeight()` - sets the height of rows in the list
  - `setSelectionBG()` - sets the URL for the image to display for a selected row
- jsx3.gui.Matrix.Column
  - `setCellBackgroundColor()` - sets the background-color of cells in the column
  - `setCellColor()` - sets the font color of cells in the column
  - `setCellFontName()` - sets the font type of cells in the column
  - `setCellFontSize()` - sets the font size of cells in the column
  - `setCellFontWeight()` - set the font weight of cells in the column
  - `setCellTextAlign()` - sets the alignment of text in cells in the column
  - `setTip()` - sets the tooltip text for cells in the column

For a complete list of available properties and parameters, see the TIBCO General Interface API Documentation.

Syntax

```
classProto.modifyMatrix = function(oMatrix, oContext)
```

Parameters

- *oMatrix* - (jsx3.gui.Matrix) A Matrix component used to display a list.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* - (string read-only) The name of the logged-in user.
  - *oContext.userGuid* - (string read-only) A unique identifier of the logged-in user.
  - *oContext.listType* - (string read-only) The view type of the list. The possible values are:
  - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
  - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
  - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)
  - com.tibco.wcc.base.ListContainer.WORKITEM (Work Items)
  - com.tibco.wcc.base.ListContainer.PROCESSINSTANCE (Process Instances)
  - com.tibco.wcc.base.ListContainer.AUDITEVENT (Events)
  - com.tibco.wcc.base.ListContainer.PROCESSTEMPLATES (Process Templates list when starting a process instance)

  - com.tibco.wcc.base.ListContainer.PROCESSTEMPLATESEX (Process Templates list when creating a process view)
  - com.tibco.wcc.base.ListContainer.BSVIEW (Business Services)

Returns

None

modifyViewListData

This method provides a means of modifying display values for work items in work view, process view, and event view lists. The XML that populates the list is passed to this method, providing a means to manipulate list data prior to it being displayed. Text values of view names and descriptions may be changed, allowing for localization or customization, plus items in the list may be reordered.

- Care must be taken when changing the display text of work items in the work views list. The "Inbox" view is a special application view that is already localized to the current language setting. Changing "Inbox" to any other value can result in unpredictable results.
- You must also be careful when reordering items in the work views list. Work items grouped under "My Work" and "Supervised Work" may be reordered within the groups, but must remain within their original group to prevent application failures.

This method is called when the list is first constructed, and every time it is refreshed.

Syntax

```
classProto.modifyViewListData = function(oValue, oContext)
```

Parameters

- *oValue.listXML* - (jsx3.xml.CDF.Document) An XML document that populates the list. Contains a <data> root node that is the parent of zero or more <record> nodes, which represent the items in the list.
- *oContext* - (Object) An object that provides information about the list being modified with the following properties:
  - *oContext.userName* (string read-only) - The name of the logged-in user.
  - *oContext.userGuid* (string read-only) - A unique identifier of the logged-in user.
  - *oContext.listType* - (string read-only) The view type of the list. The possible values are:
    - com.tibco.wcc.base.ListContainer.WORKVIEW (Work View)
    - com.tibco.wcc.base.ListContainer.PTVIEW (Process View)
    - com.tibco.wcc.base.ListContainer.EVENTVIEW (Event View)

Returns

oValue - with listXML property containing jsx3.xml.CDF.Document to populate the list.

Callout Utilities

The WCC base application provides some tools and “helper” methods that can be used in conjunction with the callout interface methods.

The available callout utilities are summarized below:

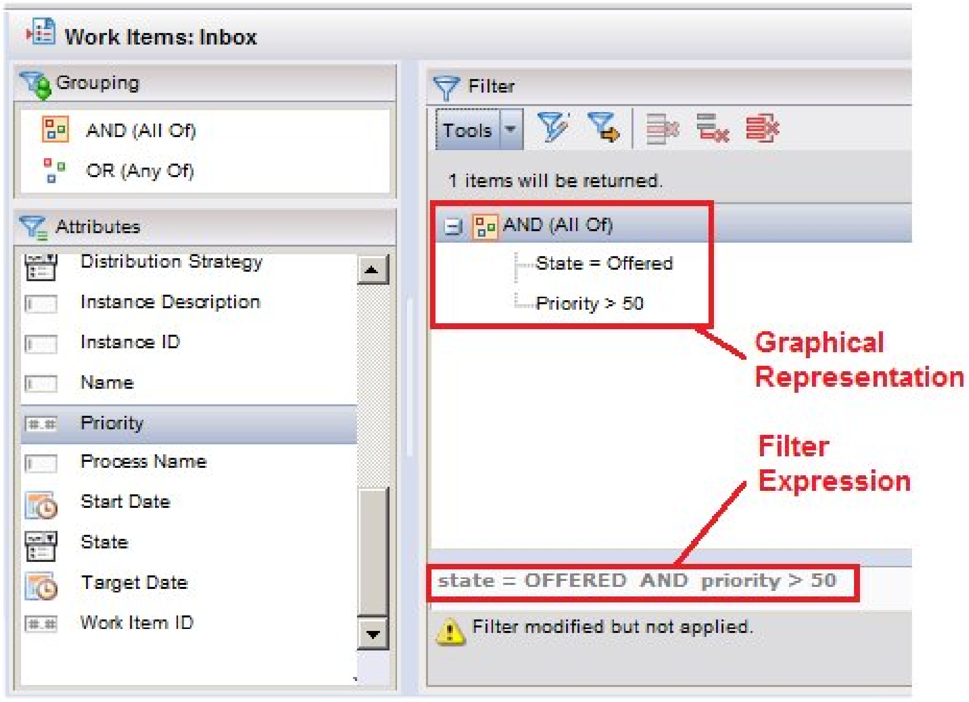
- Generating a Filter Expression - From within a WCC application you can view and obtain filter expression syntax, as well as the XML that graphically represents the filter expression. These can then be passed in callout methods to create a new filter or to modify an existing filter by appending an additional expression.

- appendFilterExpression - This method appends a specified filter expression string to the original filter expression returned in the `oValue` parameter of the filter override methods.
- appendFilterXML - This method appends an XML string, that graphically represents a filter expression in the Filter dialog, to the original graphical filter XML returned in the `oValue` parameter of the filter override methods.
- getLoggedInUserDetail - This method returns an object containing details of the organizational status of the currently logged-in user, including the groups and positions the user is a member of, as well as the privileges associated with the user.
- getUserDetail - This method returns an object containing details of the organizational status of the specified user, including the groups and positions the user is a member of, as well as the privileges associated with the user.
- getDisplayedListItems - These methods return properties for the items either displayed or selected in various types of lists.
- doRefresh - This method is used to update a list, including execution of any callout methods, which may update the list filter, sort, or columns.

Generating a Filter Expression

There are two components to a filter definition: the actual filter expression and the XML that represents the graphical display of the filter on the Filter dialog in a WCC application.

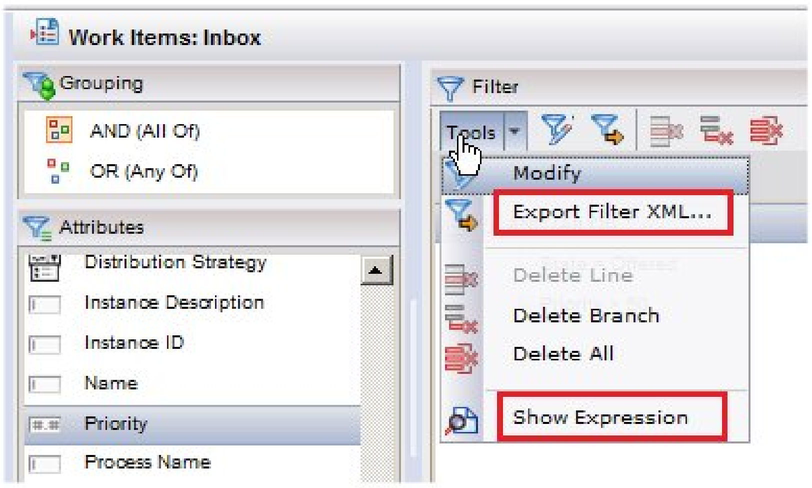
For example:



Some of the callout interface methods (specifically, overrideBaseFilter, overrideFilterOnLoad and overrideFilterOnRefresh) allow you to pass a filter expression to create a new filter, or to append it to an existing filter. The overrideBaseFilter and overrideFilterOnLoad methods also allow you to pass the XML that represents the graphical filter expression so that the user can see a graphical image of the filter (the filter generated by the overrideFilterOnRefresh method is not intended to be viewed by the user).

You can obtain the filter expression and graphical filter XML by defining a filter on the Filter dialog in a WCC application.

When a filter is defined, the filter expression syntax is displayed in the Filter dialog when you select **Show Expression** on **Tools** menu. The XML that represents that graphical display is produced by selecting **Export Filter XML...** on the **Tools** menu.



Filter expressions can also just be written manually, if desired. For information about creating filter expressions, see the “Sorting List" and "Filtering Lists” topics in the *TIBCO ActiveMatrix BPM Developer’s Guide*.

The only way to obtain the XML that represents the graphical filter display is by using the **Export Filter XML...** function in the WCC application.

Generating the Graphical Filter XML

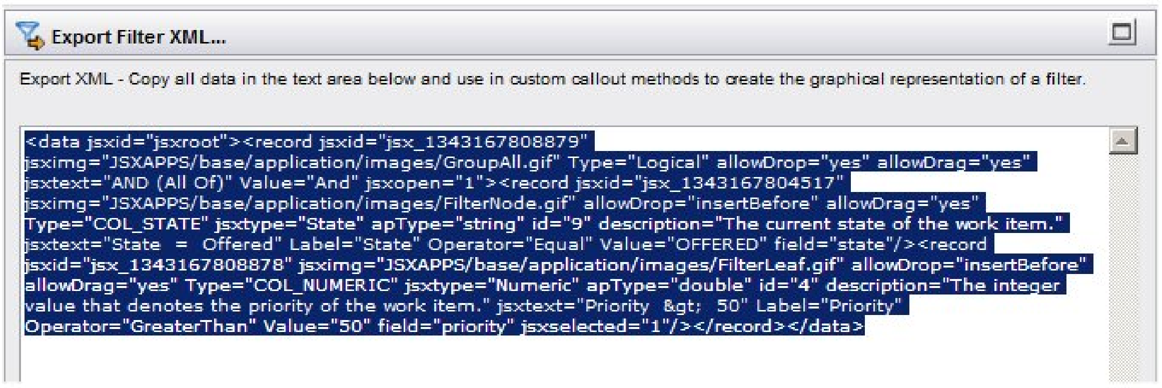
You can generate the XML that displays the graphical representation of a filter expression.

Procedure

1. Open the Filter dialog and define the filter that is to be added by the callout method.
2. From the **Tools** menu, select **Export Filter XML...**.

   A dialog is displayed that contains the XML that represents the filter expression.

   Copy all of the text in the dialog:



Export Filter XML...

Export XML - Copy all data in the text area below and use in custom callout methods to create the graphical representation of a filter.

```
<data jsxid="jsxroot"><record jsxid="jsx_1343167808879"
jsximg="JSXAPPS/base/application/images/GroupAll.gif" Type="Logical" allowDrop="yes" allowDrag="yes"
jsxtext="AND (All Of)" Value="And" jsxopen="1"><record jsxid="jsx_1343167804517"
jsximg="JSXAPPS/base/application/images/FilterNode.gif" allowDrop="insertBefore" allowDrag="yes"
Type="COL_STATE" jsxtype="State" apType="string" id="9" description="The current state of the work item."
jsxtext="State  =  Offered" Label="State" Operator="Equal" Value="OFFERED" field="state"/><record
jsxid="jsx_1343167808878" jsximg="JSXAPPS/base/application/images/FilterLeaf.gif" allowDrop="insertBefore"
allowDrag="yes" Type="COL_NUMERIC" jsxtype="Numeric" apType="double" id="4" description="The integer
value that denotes the priority of the work item." jsxtext="Priority  &gt;  50" Label="Priority"
Operator="GreaterThan" Value="50" field="priority" jsxselected="1"/></record></data>
```

3. Use the copied XML in either the overrideBaseFilter or overrideFilterOnLoad method to produce a graphical display of the filter.

Result

Also see appendFilterXML for information about appending the XML from the **XML Filter XML...** function to existing graphical filter XML.

appendFilterExpression

This method appends a specified filter expression string to the original filter expression returned in the `oValue` parameter of the filter callout methods. This filter is an expression string that is submitted to the server to filter the list.

Syntax

```
this.app.appendFilterExpression(filterExpression, appendExpression, operator)
```

Parameters

- *filterExpression* - (string) The original expression
- *appendExpression* - (string) The filter expression to be appended to the original
- *operator* - (string) The logical operator to use when appending the expression: "AND" or "OR".

Returns

A string containing the resulting filter expression.

appendFilterXML

This method appends an XML string, that graphically represents a filter expression in the Filter dialog, to the original graphical filter XML returned in the `oValue` parameter of the filter callout methods.

Syntax

```
this.app.appendFilterXML(filterXML, appendXML)
```

Parameters

- *filterXML* - (string) The original XML.
- *appendXML* - (string) The XML to be appended to the original XML. This is obtained using the **XML Filter XML...** function (see Generating a Filter Expression).

Returns

A string containing the resulting graphical filter XML string used to graphically represent the filter expression in the Filter dialog.

getLoggedInUserDetail

This method returns an object containing details of the organizational status of the currently logged in user, including the groups and positions the user is a member of, as well as the privileges associated with the user.

Syntax

```
this.app.getLoggedInUserDetail(version, forceRefresh)
```

Parameters

- *version* - (string) The version of the organization model from which to obtain the user details (-1 for the latest version).
- *forceRefresh* - (boolean) When the **getLoggedInUserDetail** method is executed, a request is sent to the server to obtain user details and the results are cached locally. This avoids unnecessary subsequent server requests that would likely obtain the same results. Setting *forceRefresh* to true causes any cached results to be discarded and a new server request to be issued.

Returns

- oDetail - (Object) Provides detailed information about the logged-in user with the following properties:
  - oDetail.guid - (string) A unique identifier for the resource.
  - oDetail.name - (string) The name of the resource.
  - oDetail.containerId - (string) ID of the container used to create the resource.
  - oDetail.containerName - (string) The name of the container used to create the resource.
  - oDetail.groups - (object) Contains a property for each group the resource is a member (see below or additional information).
  - oDetail.positions - (object) Contains a property for each position the resource holds (see below or additional information).
  - oDetail.privileges - (object) Contains a property for each privilege the resource holds (see below or additional information).
  - oDetail.capabilities - (object) Contains a property for each capability specified for the resource (see below or additional information).
  - oDetail.attributes - (object) Contains a property for each attribute specified for the resource (see below or additional information).
  - oDetail.pushDestinations - (object) Contains a property for each push destination set up for the resource (see below or additional information).

All of the objects above are used to hold a collection of similar subordinate objects. The property name used to reference the subordinate object is also used as the name of that object. For instance, if the resource is a member of a group named CustomerService, there is a property named `oDetail.groups.CustomerService` that is an object, where:

```
oDetail.groups.CustomerService.name = 'CustomerService';
```

Each Object under **oDetail.groups** has the following properties:

- guid - (string) unique identifier
- name - (string) unique name
- label - (string) display name (may not be unique)
- startDate - (date) date when membership takes effect
- endDate - (date) date when membership expires

Each Object under **oDetail.positions** has the following properties:

- guid - (string) unique identifier
- name - (string) unique name

- label - (string) display name (may not be unique)
- startDate - (date) date when membership takes effect
- endDate - (date) date when membership expires

Each Object under **oDetail.privileges** has the following properties:

- guid - (String) unique identifier
- name - (String) unique name
- label - (String) display name (may not be unique)
- values - (Object []) Array of objects, each representing one way in which the privilege has been granted. The privilege may be granted through any number of memberships with an optional qualifying value.
  - value - (String) an optional value that qualifies the privilege
  - originGuid - (String) unique identifier of position or group granting the privilege
  - originName - (String) unique name for the position or group
  - originLabel - (String) display name for the position or group (may not be unique)
  - originType - (String) either 'POSITION' or 'GROUP'

You can verify a privilege has been granted in some way (any value) in this way:

```
var bPrivilegeHeld = oDetail.privileges.SecureAccess != null;
```

To check for a specific value you will need to loop through the array of Values:

```
var bPrivilegeHeld = false;
   for (var x=0; x<oDetail.privileges.SecureAccess.values[x]; x++) {
      if (oDetail.privileges.SecureAccess.values[x]=='Level9') {
         bPrivilegeHeld = true;
         break;
      }
   }
```

To verify a specific privilege with a specific value is granted through a specific membership:

```
var bPrivilegeHeld = false;
   for (var x=0; x<oDetail.privileges.SecureAccess.values.length; x++) {
      var oItem = oDetail.privileges.SecureAccess.values[x]
      if (oItem .Value=='Level9' && oItem.OriginName=='ChiefOfSecurity') {
         bPrivilegeHeld = true;
         break;
      }
   }
```

Each Object under **oDetail.capabilities** has the following properties:

- guid - (string) unique identifier
- name - (string) unique name
- label - (string) display name (may not be unique)
- dataType - (string) type of data held in the Value property
- value - (various data types) value specified for the capability. These are the DataTypes followed by the JavaScript type used to represent them. When DataType is blank the Value will be null.
  - 'Boolean' - (boolean)
  - 'Date' - (date) the time information should be ignored
  - 'DateTime' - (date)
  - 'Time' - (date) the date information should be ignored

  - 'Enum' - (string)
  - 'EnumSet' - (string [])
  - 'Text' - (string)
  - 'Integer' - (string)
  - 'Decimal' - (string)

Each Object under **oDetail.attributes** has the following properties:

- guid - (string) unique identifier
- name - (string) unique name
- label - (string) display name (may not be unique)
- dataType - (string) type of data held in the Value property
- value - (various data types) value specified for the capability. These are the DataTypes followed by the JavaScript type used to represent them. When DataType is blank the Value will be null.
  - 'Boolean' - (boolean)
  - 'Date' - (date) the time information should be ignored
  - 'DateTime' - (date)
  - 'Time' - (date) the date information should be ignored
  - 'Enum' - (string)
  - 'EnumSet' - (string [])
  - 'Text' - (string)
  - 'Integer' - (string)
  - 'Decimal' - (string)

Each Object under **oDetail.pushDestinations** has the following properties:

- enabled - (boolean) true if work items directed at this channel should be pushed to the user
- channelType - (string) type of push destination (at this time, only "EmailChannel" is used)
- channelId - (string) identifier for the channel
- target - (string) information used to address the work to the user, (at this time, only email addresses are used). Note that the target could be a unique value that was entered for this push destination, or it could be a value retrieved from a resource attribute. If it comes from a resource attribute, the following three properties will also contain information:
- attributeGuid - (string) GUID of the resource attribute
- attributeName - (string) unique name of the resource attribute
- attributeLabel - (string) display name of the resource attribute (may not be unique)

getUserDetail

This method returns an object containing details of the organizational status of the specified resource, including the groups and positions the user is a member of, as well as the privileges associated with the user.

Syntax

```
this.app.getUserDetail(version, resourceGuid)
```

Parameters

- *version* - (string) The version of the organization model from which to obtain the user details (-1 for the latest version).
- *resourceGuid* - (string) A unique identifier for the resource.

Returns

oDetail - (Object) Provides detailed information about the specified resource. The format of this object is identical to the one described for getLoggedInUserDetail.

getDisplayedListItems

The method returns an array of objects, each representing one item currently displayed in the list.

This method can be used with the following list types:

- Work Views (‘wccWorkViews’)
- Work Items (‘wccWorkItems’)
- Process Views (‘wccProcessViews’)
- Process Instances (‘wccProcessInstances’)
- Business Services (‘wccBusinessServices’)
- Event Views (‘wccEventViews’)
- Event Viewer (‘wccEventViewer’)
- Process Templates (‘wccProcessTemplates’)
- Process Templates Ex (‘wccProcessTemplatesEx’)

This method is called from the list container object whose displayed-item information you want returned. An example is shown below:

```
var oWorkItems = this.app.getAppBlock().getDescendantOfName('wccWorkItems');
oWorkItems.doRefresh();
var oDisplayedWorkItems = oWorkItems.getDisplayedListItems();
```

Also see doRefresh.

The **getDisplayedListItems** method returns an array of objects, each one representing an item displayed in the list. The properties on each object depends on the list type, as follows:

- **Work Views**:
  - oItem.view - name of the view
  - oItem.description - description for the view
  - oItem.type - either "UserEdit" or "Default"
  - oItem.parentNode - either "MYWORK" or "SUPERVISEDWORK"
  - oItem.filter - base filter associated with the work item list displayed for this view
  - oItem.temporary - (optional) "true" if it is a temporary list; not preserved after logout
  - oItem.entityType - (optional) for supervised lists, this indicates the type of organizational entity associated with the work list (RESOURCE, GROUP, or POSITION)
  - oItem.guid - (optional) for supervised lists, the unique identifier for the organizational entity
  - oItem.modelVersion - (optional) for supervised lists, the model version of the organizational entity

  - oItem.SupervisedWorkItemState - (optional) for supervised lists, the state of the items listed, either "Offered" or "Allocated"
- **Work Items**:
  - oItem.id
  - oItem.version
  - oItem.description
  - oItem.startDate
  - oItem.endDate
  - oItem.distributionStrategy
  - oItem.priority
  - oItem.groupId
  - oItem.activityId
  - oItem.appId
  - oItem.appInstance
  - oItem.appName
  - oItem.scheduleStatus
  - oItem.state
  - oItem.workTypeId
  - oItem.workTypeUid
  - oItem.workTypeVersion
  - oItem.workTypeDescription
  - oItem.attribute1
  - oItem.attribute2
  - oItem.attribute3
  - oItem.attribute4
  - oItem.attribute5
  - oItem.attribute6
  - oItem.attribute7
  - oItem.attribute8
  - oItem.attribute9
  - oItem.attribute10
  - oItem.attribute11
  - oItem.attribute12
  - oItem.attribute13
  - oItem.attribute14
  - oItem.attribute15
  - oItem.attribute16
  - oItem.attribute17
  - oItem.attribute18

- oItem.attribute19
- oItem.attribute20
- oItem.attribute21
- oItem.attribute22
- oItem.attribute23
- oItem.attribute24
- oItem.attribute25
- oItem.attribute26
- oItem.attribute27
- oItem.attribute28
- oItem.attribute29
- oItem.attribute30
- oItem.attribute31
- oItem.attribute32
- oItem.attribute33
- oItem.attribute34
- oItem.attribute35
- oItem.attribute36
- oItem.attribute37
- oItem.attribute38
- oItem.attribute39
- oItem.attribute40

- **Business Services**:

  Note that business services in the list are only retrieved when the categories headers are opened. Keep that in mind when retrieving the displayed list items. Also, the list may include duplicates when a service is returned as part of a query or is one of the recent items.

  - oItem.processName
  - oItem.description
  - oItem.modelName
  - oItem.version

- **Process Views**:

  - oItem.view
  - oItem.description
  - oItem.type - "UserEdit" or "Default"
  - oItem.templateIds

- **Process Instances**:

  - oItem.instanceStatus
  - oItem.instanceVersion

  - oItem.instanceName
  - oItem.id
  - oItem.moduleName
  - oItem.instanceWaitingWorkCount
  - oItem.instancepriority
  - oItem.instanceParentProcessId
  - oItem.instanceStartDate
  - oItem.instanceActivityFaultData
  - oItem.instanceActivityFaultName
  - oItem.instanceFailedActivityName
  - oItem.* - (optional) Any custom attributes that are displayed for the list are also included. Note that for an attribute to be displayed, it must exist in all of the processes included in the list.
- **Event Views**:
  - oItem.view
  - oItem.descriptionType - "summary" or "detail"
  - oItem.filterId - "Summary" or "All"
  - oItem.viewType - "context" or …
  - oItem.description
  - oItem.filter
- **Event Viewer**:
  - oItem.* - There are a variety of properties for items returned by the event viewer, with different event types returning different subsets of data. To find the specific property name used for data, you could examine what is returned for an entry that includes it. Or you can build a filter that includes the attribute, then without actually saving the filter, choose to show the expression and see the name that is used in that expression.
- **Process Templates and Process Templates Ex**:
  - oItem.processName
  - oItem.description
  - oItem.moduleName
  - oItem.version
  - oItem.priority
  - oItem.creationDate

  Note that the data for these lists includes entries for the specific versions of processes.

  For the Process Template Ex lists there are also entries that are not version specific for each process. The only value specified in these entries is the processName. Also, this list type is presented as a tree with check boxes, so when retrieving the selected items, it returns all checked entries.

getSelectedListItems

The method returns an array of objects, each representing one item currently selected in the list.

This method can be used with the following list types:

- Work views (‘wccWorkViews’)
- Work items (‘wccWorkItems’)
- Process views (‘wccProcessViews’)
- Process instances (‘wccProcessInstances’)
- Business services (‘wccBusinessServices’)
- Event Views (‘wccEventViews’)
- Event Viewer (‘wccEventViewer’)
- Process Templates (‘wccProcessTemplates’)
- Process Templates Ex (‘wccProcessTemplatesEx’)

This method is called from the list container object whose selected-item information you want returned. An example is shown below:

```
var oWorkItems = this.app.getAppBlock().getDescendantOfName('wccWorkItems');
oWorkItems.doRefresh();
var oDisplayedWorkItems = oWorkItems.getSelectedListItems();
```

Also see doRefresh.

The **getSelectedListItems** method returns an array of objects, each one representing an item selected in the list. The properties on each object depends on the list type. The properties returned are the same as those returned from the getDisplayedListItems method.

doRefresh

This method updates the current list, including the execution of any callouts which may update the filtering, sorting, or columns.

If some other part of the application is used to determine the filtering, sorting, or columns of a list, ideally, changes to that input would trigger an update of that list. Otherwise, the changes would only occur if the user explicitly refreshes the list. The **doRefresh** method can be used to trigger that update from other parts of the application.

For instance, a callout for a process instance list could set the filter so that it shows only items for processes referenced by the currently selected work item(s). In code triggered by a change in the work item selection, the **doRefresh** method could be called on the process instance list, which will then trigger the application of the new filter.

- Work views (‘wccWorkViews’)
- Work items (‘wccWorkItems’)
- Process views (‘wccProcessViews’)
- Process instances (‘wccProcessInstances’)
- Business services (‘wccBusinessServices’)
- Event Views (‘wccEventViews’)
- Event Viewer (‘wccEventViewer’)
- Process Templates (‘wccProcessTemplates’)
- Process Templates Ex (‘wccProcessTemplatesEx’)

This method is called from the list container object you want to refresh. An example of its use is shown below:

```
var oWorkItems = this.app.getAppBlock().getDescendantOfName('wccWorkItems');
oWorkItems.doRefresh();
var oDisplayedWorkItems = oWorkItems.getSelectedListItems();
```

Workspace Attributes

The filterability/sortability of work item attributes is controlled by the corresponding ActiveMatrix BPM server component that defines the attribute, not by TIBCO Workspace.

The filterability/sortability is included in this document for convenience of reference.

Work Item Attributes

This topic lists all attributes available as display columns in views of work items, and identifies which attributes may be used to sort and filter the work view.

The case of the **id** property value differs for some attributes when using the overrideColumnsOnLoad , overrideColumnsOnRefresh , and overrideColumnAttributes callout methods. When referencing the **id** property, use the attribute **id** in parentheses for the column callouts. For filter and sort callouts, use the **id** that starts with a lowercase character.

id	Label / Header	Filterable	Sortable
StateImage	State Image		
appName (AppName)	Process Name	Yes	Yes
Description	Description	Yes	
appInstanceDescription (AppInstanceDescription)	Instance Description	Yes	Yes
appInstance (AppInstance)	ID	Yes	Yes
priority (Priority)	Work Item Priority	Yes	Yes
startDate (StartDate)	Start Date	Yes	Yes
DeadlineImage	Deadline		
endDate (EndDate)	Target Date	Yes	Yes
distributionStrategy (DistributionStrategy)	Distribution Strategy	Yes	Yes
State	State	Yes	
ActivityID	Activity ID		
ScheduleStatus	Schedule Status		
Version	Work Item Version		
GroupID	Group ID		
id (ID)	Work Item ID	Yes	Yes
Name	Name	Yes	

id	Label / Header	Filterable	Sortable
Visible	Visibility		
attribute1 (Attribute1)	Attribute #1	Yes	Yes
attribute2 (Attribute2)	Attribute #2	Yes	Yes
attribute3 (Attribute3)	Attribute #3	Yes	Yes
attribute4 (Attribute4)	Attribute #4	Yes	Yes
attribute5 (Attribute5)	Attribute #5	Yes	Yes
attribute6 (Attribute6)	Attribute #6	Yes	Yes
attribute7 (Attribute7)	Attribute #7	Yes	Yes
attribute8 (Attribute8)	Attribute #8	Yes	Yes
attribute9 (Attribute9)	Attribute #9	Yes	Yes
attribute10 (Attribute10)	Attribute #10	Yes	Yes
attribute11 (Attribute11)	Attribute #11	Yes	Yes
attribute12 (Attribute12)	Attribute #12	Yes	Yes
attribute13 (Attribute13)	Attribute #13	Yes	Yes
attribute14 (Attribute14)	Attribute #14	Yes	Yes
attribute15 (Attribute15)	Attribute #15	Yes	Yes
attribute16 (Attribute16)	Attribute #16	Yes	Yes
attribute17 (Attribute17)	Attribute #17	Yes	Yes
attribute18 (Attribute18)	Attribute #18	Yes	Yes
attribute19 (Attribute19)	Attribute #19	Yes	Yes
attribute20 (Attribute20)	Attribute #20	Yes	Yes
attribute21 (Attribute21)	Attribute #21	Yes	Yes
attribute22 (Attribute22)	Attribute #22	Yes	Yes
attribute23 (Attribute23)	Attribute #23	Yes	Yes
attribute24 (Attribute24)	Attribute #24	Yes	Yes
attribute25 (Attribute25)	Attribute #25	Yes	Yes

id	Label / Header	Filterable	Sortable
attribute26 (Attribute26)	Attribute #26	Yes	Yes
attribute27 (Attribute27)	Attribute #27	Yes	Yes
attribute28 (Attribute28)	Attribute #28	Yes	Yes
attribute29 (Attribute29)	Attribute #29	Yes	Yes
attribute30 (Attribute30)	Attribute #30	Yes	Yes
attribute31 (Attribute31)	Attribute #31	Yes	Yes
attribute32 (Attribute32)	Attribute #32	Yes	Yes
attribute33 (Attribute33)	Attribute #33	Yes	Yes
attribute34 (Attribute34)	Attribute #34	Yes	Yes
attribute35 (Attribute35)	Attribute #35	Yes	Yes
attribute36 (Attribute36)	Attribute #36	Yes	Yes
attribute37 (Attribute37)	Attribute #37	Yes	Yes
attribute38 (Attribute38)	Attribute #38	Yes	Yes
attribute39 (Attribute39)	Attribute #39	Yes	Yes
attribute40 (Attribute40)	Attribute #40	Yes	Yes

Process Instance Attributes

This topic lists all attributes available as display columns in views of process instances.

The table below identifies which attributes may be used to sort and filter the view. Note that the list only shows system attributes; custom process attributes may also be defined, but are not shown in this list.

id	Label / Header	Filterable	Sortable
INSTANCE.ACTIVITY_FAULT_DATA(1)	Activity Fault Data	Yes	Yes
INSTANCE.ACTIVITY_FAULT_NAME1	Activity Fault Name	Yes	Yes
INSTANCE.FAILED_ACTIVITY_NAME1	Failed Activity Name	Yes	Yes
INSTANCE.ID	ID	Yes	Yes
INSTANCE.NAME	Process Name	Yes	Yes
INSTANCE.PARENT_PROCESS_ID	Parent Process ID	Yes	Yes

id	Label / Header	Filterable	Sortable
INSTANCE.PRIORITY	Priority	Yes	Yes
INSTANCE.START_DATE	Start Date	Yes	Yes
INSTANCE.STATUS	Status	Yes	Yes
INSTANCE.VERSION	Version	Yes	Yes
INSTANCE.WAITING_WORK_COUNT	Outstanding Count	Yes	Yes
MODULE.NAME	Module Name	Yes	Yes

(1) Only applicable to the special-use halted process instance lists. For information, see "Halted Process Instances" in the *TIBCO Workspace User's Guide*.

Event Attributes

This topic lists all attributes available as display columns in views of events.

The table below identifies which attributes may be used to sort and filter the view.

id	Label / Header	Filterable	Sortable
entities	Resource entity GUID	Yes	Yes
fieldName	Field name		
guid	GUID		
scriptTypeUid	Script type GUID		
workGroupDesc	Work group description		
workGroupId	Work group ID		
workGroupType	Work group type		
workItemData	Work item data		
workItemScheduleEnd	Work item schedule target date	Yes	Yes
workItemScheduleStart	Work item schedule start date	Yes	Yes
workTypeDesc	Work type description		
workTypeVersion	Work type version		
applicationActivityInstanceId	Application activity instance ID	Yes	Yes
applicationActivityModelId	Application activity model ID	Yes	Yes

id	Label / Header	Filterable	Sortable
applicationActivityName	Application activity name	Yes	Yes
applicationId	Application ID		
applicationName	Application name	Yes	Yes
entityId	Entity ID	Yes	Yes
entityName	Entity name		
entityType	Entity type	Yes	Yes
extendedMessage	Extended message		
managedObjectDescription	Managed object description		
managedObjectId	Managed object ID	Yes	Yes
managedObjectName	Managed object name	Yes	Yes
managedObjectStatus	Managed object status	Yes	Yes
managedObjectType	Managed object type	Yes	Yes
managedObjectUrl	Managed object URL		
managedObjectVersion	Managed object version	Yes	Yes
parentObjectId	Parent object ID	Yes	Yes
resourceId	Resource ID	Yes	Yes
resourceName	Resource name	Yes	Yes
sequenceId	Sequence ID		
systemActionId	System action ID	Yes	Yes
workModelId	Work model ID		
workTypeId	Work type ID		
dacAllTimeIsWorkingTime	All time working time		
dacCalendarEntryGuid	Calendar entry GUID		
dacFreeBusyDuration	Free/busy duration		
dacFreeBusyEnd	Free/busy end time		
dacFreeBusyStart	Free/busy start time		

id	Label / Header	Filterable	Sortable
dacFreeBusyType	Free busy type		
dacRequestXml	DAC request xml		
dacResponseXml	DAC response xml		
parameterName	Parameter name		
ruleValue	Rule value		
actionPrivilege	System action privilege		
allocationMethod	Allocation method		
assocQualifier	Association qualifier		
businessAttribute	Business attribute		
char	Character		
childNumber	Child number		
children	Children		
containerCount	LDAP container count		
containerDescription	LDAP container description		
containerMappingString	LDAP container mapping string		
defaultValue	Default business parameter value		
deploymentMessage	Deployment message		
deploymentUrl	Deployment URL		
entitiesFound	Entities found		
entitiesToProcess	Entities to process		
entity	Entity		
expressionsFound	Expressions found		
fileUrl	External file URL		
filterElement	Filter element		
filterPrivilegeElement	Filter privilege element		
filterQualifierElement	Filter qualifier element		

id	Label / Header	Filterable	Sortable
firstExecution	First execution		
foundEscape	Found escape		
foundMetaDataType	Found meta data type		
foundWildcard	Found wildcard		
immediateChildrenOnly	Immediate children only		
index	Generic index		
isPrimary	Primary LDAP resource		
isValid	Valid value		
lastAccess	Last LDAP access		
ldapAlias	LDAP alias		
ldapAttribute	LDAP attribute		
ldapAttributeValue	LDAP attribute value		
ldapContainer	LDAP container		
ldapDisplayAttributes	LDAP resources/attributes		
ldapDisplayFields	LDAP display fields		
ldapDn	LDAP DN		
ldapFilter	LDAP filter		
ldapResourceId	LDAP resource ID	Yes	Yes
ldapResources	LDAP resources		
ldapSearchResultList	LDAP search results		
ldapSearchString	LDAP search string		
ldapSource	LDAP source		
localValue	Local value		
localValueN	Normalized local value		
majorVersion	Major version		
microVersion	Micro version		

id	Label / Header	Filterable	Sortable
minorVersion	Minor version		
modifiedCount	Modified count		
name	Name		
node	Node		
numberOfChildren	Number of children		
orgModelId	Organization model ID		
originalQueryValue	Original query value		
paramDescriptor	Parameter descriptor GUID		
parameterDescription	Parameter descriptor description		
parameterType	Parameter type		
pushDestination	Push destination	Yes	Yes
pushDestinationNumTargets	Push destination target count		
pushDestinationTarget	Push destination target	Yes	Yes
qualifier	Version qualifier		
qualifierDescriptor	Capability qualifier descriptor		
queryEntity	Query entity		
queryFilter	Query filter		
queryJunction	Query junction		
queryOperator	Query operator		
queryScope	Query scope		
queryString	Resource query string		
queryValue	Query value		
reason	Reason		
resultBoolean	Result boolean		
returnedDataCount	Returned data count		
returnValue	Response XML		

id	Label / Header	Filterable	Sortable
rootGroups	Root groups		
rootNode	Root node		
rootOrgUnits	Root organization units		
secondaryLdapResourceId	Secondary LDAP resource		
sourceEntity	Source entity		
stringParameter	String parameter		
stringResult	String result		
systemActionComponentId	System action component ID	Yes	Yes
systemActionDefaultValue	System action default		
targetEntity	Qualified association target		
userSettingCategory	User setting category		
userSettingId	User setting ID		
value	Value		
Version	Version		
xmlText	Request XML		
ecQueryFilter	Event collector query		
newAttributeList	New attribute list		
numberReturned	Number returned		
query	HQL query		
queryCorrelate	Correlate query		
queryGuid	Query GUID		
queryOptionsGetTotalCount	Query get total count		
queryOptionsNumberOfItems	Query item count		
queryOptionsStartPosition	Query start position		
queryTag	Query tag		
registeredAttributeName	Registered attribute name		

id	Label / Header	Filterable	Sortable
registeredComponentId	Registered component ID		
registeredComponentName	Registered component name		
registeredComponentVersion	Registered component version		
searchTag	Search tag		
componentClassName	Component class name		
componentId	Component ID	Yes	Yes
compositeApplicationName	Composite application name		
contextId	Context ID	Yes	Yes
correlationId	Correlation ID	Yes	Yes
creationTime	Creation time	Yes	Yes
environmentName	Environment name		
eventType	Event type		
hostAddress	Host address	Yes	Yes
hostName	Host name	Yes	Yes
id	Event ID	Yes	Yes
lineNumber	Line number		
message	Message	Yes	Yes
messageCategory	Message category	Yes	Yes
messageId	Message ID	Yes	Yes
methodId	Method ID		
methodName	Method name		
nodeName	Node name	Yes	Yes
parentContextId	Parent context ID	Yes	Yes
preciseCreationTime	Precise creation time		
principalDomain	Principal domain		
principalId	Principal ID	Yes	Yes

id	Label / Header	Filterable	Sortable
principalName	Principal name	Yes	Yes
priority	Priority	Yes	Yes
requestReceived	Request received		
serviceName	Service name		
severity	Severity	Yes	Yes
severityImage	Severity Image		
stackTrace	Stack trace		
stats	Statistics		
threadId	Thread ID		
threadName	Thread name		
authorization	Authorization		
diagnostic	Diagnostic info		
host	Host		
messageText	Message text		
referer	Referer		
sessionCreationTime	Session creation time		
sessionId	Session ID		
sessionLastAccessedTime	Session last accessed time		
sessionMaxActiveInterval	Session maximum active interval		
userAgent	User-Agent		
bundleClassLoader	Bundle class loader		
pfeRequestXml	Request XML string		
pfeResponseXml	Response XML string		
activityId	Activity ID		
activityName	Activity name		
faultDetails	Fault details		

id	Label / Header	Filterable	Sortable
modelId	XPDL ID		
moduleName	Module name	Yes	Yes
moduleVersion	Module version		
parentActivityInstanceId	Parent activity instance ID	Yes	Yes
parentProcessInstanceId	Parent process instance ID	Yes	Yes
priorStepId	Prior step ID	Yes	Yes
processDetails	Process instance details	Yes	Yes
processId	Process instance ID		
processName	Process template name		
processPriority	Process instance priority	Yes	Yes
subprocessInstanceId	Subprocess instance ID	Yes	Yes
subprocessName	Subprocess name	Yes	Yes
subprocessTemplateId	Subprocess template ID		
subprocessVersion	Subprocess version	Yes	Yes
baseDeployDirectory	Base deployment directory		
dbErrorMsg	Failed database query		
deployUrl	Deployed artifact URL		
httpExtenderName	HTTP extender name		
httpRequestAttName	HTTP binding attributes		
parameters	Presentation parameters		
pushWorkId	Push work ID		
pushWorkMode	Push work mode		
pushWorkMsg	Push work message		
returnMsg	Undeployment message		
serviceArchive	Service archive		
serviceArchiveId	Service archive ID		

id	Label / Header	Filterable	Sortable
serviceArchiveLocation	Service archive location		
serviceArchiveName	Service archive name		
workItemPayload	Work item payload		
workTypeFileName	Work type file name		
workTypeFilePath	Work type file path		
workTypeUnitId	Work type unit ID		
workTypeUnitName	Work type unit name		
wpMethodName	Work presentation method name		
actionType	Action type		
baseDeployLocation	Base deployment location		
channelId	Channel ID	Yes	Yes
channelType	Channel type		
directory	Directory		
workTypeResource	Work type resource		
description	Description		
mailHostFile	Host mail file		
classLoader	Class loader		
formBaseUrl	Form base URL		
pilingLimit	Piling limit		
threadContextClassLoader	Thread context class loader		
path	Composite path		
Links	Event Links		
genericLogMessage	Generic log message		
processTime	Request process time		
requestXML	Request XML		
responseXML	Response XML		

Using WCC Components in Mash-Ups

WCC components can be included with *non-WCC code* in mash-up applications. The non-WCC code and WCC components communicate via the TIBCO PageBus mechanism.

The term *non-WCC code* is used to refer to the part of a mash-up application that is non-WCC.

The following can be accomplished in a mash-up application when using both non-WCC code and WCC components:

- Non-WCC code can subscribe to an event published by a WCC component, then consume the data in the event payload.

  For more information, see Non-WCC Subscribing to WCC Events.
- Non-WCC code can publish an event that causes one of the following to be displayed:
  - work item list
  - process instance list
  - event list (Event Viewer)

    For more information, see Non-WCC Code Publishing Events.

When the non-WCC code and WCC components are contained within one iframe, the standard PageBus mechanism can be used.

TIBCO PageBus Managed Hub

If the non-WCC code and WCC component are isolated in separate iframes in the mash-up application, the **TIBCO PageBus Managed Hub** must be used to subscribe or publish events across the iframe boundary.

For information about the PageBus Managed Hub, see the *TIBCO PageBus Developer's Guide*.

To assist you in accomplishing this, an **IframeContainer** class has been provided in Workspace that contains methods you can use when working with the PageBus Managed Hub. For information about these methods, see Using the TIBCO PageBus Managed Hub with WCC Components.

Two sample applications are also provided that illustrate the use of the methods in the **IframeContainer** class:

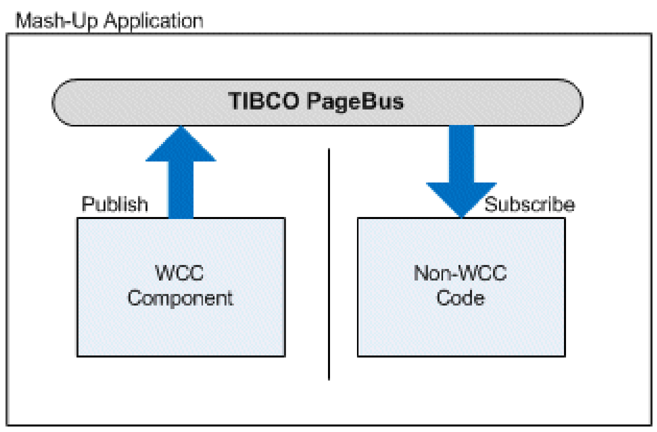
- **Login Managed Hub Application** - This sample application illustrates how to do the following when using the PageBus Managed Hub:
  - Initialize a login for a WCC application running in an iframe, by utilizing a login from a previous login session.
  - Display a WCC component (for example, the Organization Browser) in an iframe.
  - Publish an **externalShowEvents** event, which allows an external application to add a temporary event view to the event view list.

    For information, see wccLoginManagedHub Sample Application.
- **PassThru Managed Hub Application** - This sample application illustrates how to do the following when using the PageBus Managed Hub:
  - Initialize a login for a WCC application running in an iframe, by utilizing a login from a previous login session.
  - Publish a **passThruEvent** event, which passes through the TIBCO PageBus Managed Hub, where an IframeClient subscribes to the event and displays either a work item list or process instance list, depending on the payload in the published event.

For information, see wccPassThruManagedHub Sample Application.

Non-WCC Subscribing to WCC Events

When including both non-WCC code and WCC components in a mash-up application, the non-WCC code can subscribe to an event published by a WCC component, then consume the data in the event payload.



The non-WCC code must use the **Pagebus.subscribe** method to subscribe to the appropriate event. It can then consume the data that is passed in the event payload. For detailed information about the data included in event payloads for every event published by WCC components, see Component PageBus Event Payloads.

Component PageBus Event Payloads

The following sub-topics list all of the events that are published by WCC components, as well as details about the schema that represents the data that is passed in the event payload. Note that ‘schema’ in this context refers to the structure of the JavaScript object that is passed, rather than an XML schema.

These payload details are provided so that non-WCC code can subscribe to an event and consume the data as needed.

Composite Components

Composite components do not publish events themselves, although the underlying components do.

Organization / Resource Browser

Underlying components:

- Organization Browser — see Organization Browser.
- Organization Resource List — see Organization Resource List.

Process Instances Preview

Underlying components:

- Process Instances — see Process Instances.
- Preview Pane — note that the preview pane is viewed as a separate component in the context of composite components, although it cannot be installed separately. It does not publish events.

Work Items Preview

Underlying components:

- Work Items — see Work Items
- Preview Pane — note that the preview pane is viewed as a separate component in the context of composite components, although it cannot be installed separately. It does not publish events.

Data Mask Component

The Data Mask component does not publish events.

List Container Components

This topic describes the events published by the list container components.

Business Services

Event	Schema
listItemSelect	com.tibco.wcc.schema.businessServices
listExecute	com.tibco.wcc.schema.businessServices

Event Views

Event	Schema
listItemSelect	com.tibco.wcc.schema.eventViews
listItemRemoved	com.tibco.wcc.schema.eventViewsRemoved
listExecute	com.tibco.wcc.schema.eventViews

Event Viewer

Event	Schema
eventSelected	com.tibco.wcc.schema.eventSelected
attributeSelected	com.tibco.wcc.schema.attributeSelected
saveEventView	com.tibco.wcc.schema.saveEventView
saveEventViewAs	com.tibco.wcc.schema.saveEventViewAs

Process Templates

Event	Schema
listItemSelect	com.tibco.wcc.schema.processtemplates
listExecute	com.tibco.wcc.schema.processtemplates

Process Templates Ex

Event	Schema
listItemSelect	com.tibco.wcc.schema.processtemplatesex
listRefresh	com.tibco.wcc.schema.processtemplatesexrefresh

Process Views

Event	Schema
listItemSelect	com.tibco.wcc.schema.processViews
listItemRemoved	com.tibco.wcc.schema.processViewsRemoved
listExecute	com.tibco.wcc.schema.processViews

Process Instances

Event	Schema
listItemSelect	com.tibco.wcc.schema.processInstances
listExecute	com.tibco.wcc.schema.processInstances
cancelInstances	com.tibco.wcc.schema.processInstances
suspendInstances	com.tibco.wcc.schema.processInstances
resumeInstances	com.tibco.wcc.schema.processInstances
resumeHaltedInstances	com.tibco.wcc.schema.processInstances
retryInstances	com.tibco.wcc.schema.processInstances
ignoreInstances	com.tibco.wcc.schema.processInstances
showOutstanding	com.tibco.wcc.schema.processInstancesShowOutstanding
saveView	com.tibco.wcc.schema.saveProcessView
saveViewAs	com.tibco.wcc.schema.saveProcessViewAs
showEvents	com.tibco.wcc.schema.showEvents

Work Views

Event	Schema
listItemSelect	com.tibco.wcc.schema.workviews

Event	Schema
listExecute	com.tibco.wcc.schema.workviews
listItemRemoved	com.tibco.wcc.schema.workviewsRemoved

Work Items

Event	Schema
listItemSelect	com.tibco.wcc.schema.workItems
listExecute	com.tibco.wcc.schema.workItems
openItems	com.tibco.wcc.schema.workItems
openNextItem	com.tibco.wcc.schema.openNextWorkItem
cancelItems	com.tibco.wcc.schema.workItems
skipItems	com.tibco.wcc.schema.workItems
pendItems	com.tibco.wcc.schema.workItems
allocateItems	com.tibco.wcc.schema.workItems
unallocateItems	com.tibco.wcc.schema.workItems
reallocateItemsOfferSet	com.tibco.wcc.schema.workItems
reallocateItemsWorld	com.tibco.wcc.schema.workItems
showEvents	com.tibco.wcc.schema.showEvents
saveView	com.tibco.wcc.schema.saveWorkView
saveViewAs	com.tibco.wcc.schema.saveWorkViewAs

Login Component

This topic describes the events published by the login component.

Login Component

Event	Schema
loginComplete	com.tibco.wcc.schema.loginComplete

LocaleSelector Component

When a language is selected from the **LocaleSelector** component, the **setAppLocale** Application method is called, which publishes the **localeChanged** PageBus event. All WCC components inherently subscribe to the **localeChanged** event so that they can be refreshed when the locale is changed.

Note, however, the **localeChanged** event does *not* appear in the Event Editor in General Interface Builder.

For information about **setAppLocale**, see setAppLocale.

Toolbar Button Component

The Toolbar Button component does not publish events.

Note, however, if the Toolbar Button component is configured to be an Organization Browser Window Button, the underlying components displayed by that component publish events (see Organization Browser and Organization Resource List). None of the other components displayed by the Toolbar Button component publish events.

Organization Browser Components

This topic describes the events published by the Organization Browser components.

Organization Browser

Event	Schema
organizationItemSelect	com.tibco.wcc.schema.organizationbrowser
showEvents	com.tibco.wcc.schema.showEvents

Organization Resource List

Event	Schema
resourceItemSelect	com.tibco.wcc.schema.organizationresourcelist
showEvents	com.tibco.wcc.schema.showEvents

Start Instance Component

This topic describes the events published by the Start Instance component.

Start Instance Component

Event	Schema
instanceStarted	com.tibco.wcc.schema.startinstance

Event Schemas

This topic provides descriptions and examples of the data that is passed in the payload for each event schema.

Note that in every event message payload there is a **scope** property that indicates the following.

- If the **scope** property is “public”, you are free to capture that event and use the data in the payload as needed.
- If the **scope** property is “private”, it is an event that is used internally; it is subject to change or removal. Do not capture and use these events in your application.

You can view WCC component PageBus events, as well as their payload, as they fire in a WCC application using the **PageBus Event Monitor** component. For information about using this component, see Viewing Triggered Events using the PageBus Event Monitor Component .

com.tibco.wcc.schema.attributeSelected

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.attributeSelected**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message:schemaId = com.tibco.wcc.schema.attributeSelected
        scope = "public"
        Name = Name of the event attribute.
        Description = Description of the event attribute.
        Type = Type of data in the attribute: STRING, DATE, LONG, or INT.
        Value = Value stored in the attribute.
        Component = Name of the component that wrote the value to the attribute.
```

Example:

```
topic: com.tibco.wcc.order.wccPrototype.wccEventViewer.attributeSelected
message:schemaId = com.tibco.wcc.schema.attributeSelected
        scope = "public"
        Name = “componentId”
        Description = “The ID of the component generating this event, as defined
           in the Event Collector.”
        Type = “STRING”
        Value = “BX”
        Component = “N2 Logging Framework”
```

com.tibco.wcc.schema.businessServices

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.businessServices**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = com.tibco.wcc.schema.businessServices
         scope = "public"
         items = [
                   {
                     "View": Name of the business service.
                     "ProcessName": Name of the process started by the business
                         service.
                     "Description": Description of the business service.
                     "ModuleName": Path to the module containing the business
                         service.
                     "Version": Version of the business service.
                    }
                  ]
```

Example:

```
topic: com.tibco.wcc.order.wccPrototype.wccBusinessServices.listItemSelect
message: schemaId = com.tibco.wcc.schema.businessServices
         scope = "public"
         items = [
                   {
                     "View": "CSCallbackStart",
                     "ProcessName": "CSCallbackStart",
                     "Description": "CustomerService/CSCallbackStart",
                     "ModuleName": "/CSCallback/Process Packages/CSCallback.xpdl",
                     "Version": "1.0.0.201005251716"
                    }
                  ]
```

com.tibco.wcc.schema.eventSelected

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.eventSelected**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.eventSelected"
         scope = "public"
         items = [
                    {
                       "links": These define the links available from the event
                           viewer for the type of event selected, identified by the
                           messageId. These are defined in the appropriate
                           eventLinks.xml file. See the Configuring Events chapter
                           in the Workspace Configuration and Customization guide.
                       "messageId": Identifies the type of event selected.
Note - The remainder of the properties in this event message correlate to the
attributes available in the event. Attributes vary, depending on the event type. To
view the properties for a particular event type, view the event in the PageBus
Event Monitor component -- see Viewing Triggered Events using the PageBus Event
Monitor Component.
```

Example:

```
topic: com.tibco.wcc.order.wccPrototype.wccBusinessServices.listItemSelect
message: schemaId = "com.tibco.wcc.schema.eventSelected"
         scope = "public"
         items = [
                    {
                       "links": "<data jsxid=\"jsxroot\"><record jsxing=\"JSXAPPS/
                          base/application/images/WorkView.gif\" descriptionType=
                          \"summary\" defaultFilterId=\"Summary\" jsxtext=\"This
                          .
                          .
                          .
                          "severityImage": "JSXAPPS/base/application/images/
                               EventSeverityAudit.gif",
                          "correlationId": "bad15673-f3af-43bc-b42f-20c24c419a6e",
                          "creationTime": "2010-10-18 10:28:46",
                          "rawEventTimestamp": "2010-10-18T18:28:46.660+01:00",
                          "message": "Process Instance started.",
                          "detail": "Started instance of SalesCallbackProcess
                               (ID:pvm:0a123, version:1.0.0.201005251718) priority
                               is NORMAL",
                          "summary": "Started instance pvm:0a123 of
                               SalesCallbackProcess priority is NORMAL",
                          "messageId": "BX_INSTANCE_PROCESS_STARTED",
                          .
                          .
                          .
```

com.tibco.wcc.schema.eventViews

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.eventViews**.

The **eventViews** schema is used by both the **listItemSelect** and **listExecute** events on the Event Views component. A single click causes the **listItemSelect** event to be published; a double click causes both the **listItemSelect** and the **listExecute** events to be published. The **listItemSelect** event message contains properties that describe the event view. The **listExecute** event message does not contain any properties (other than **scope**), as they can be obtained from the **listItemSelect** event message (which is described below).

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = com.tibco.wcc.schema.eventViews
         scope = "public"
         items = [
                   {
                     "View": The name of the event view.
                     "Description": Event view description.
                     "DescriptionType": "detail" or "summary". These are defined in
                         the appropriate eventDescriptions.xml file. See the
                         Configuring Events chapter in the Workspace Configuration
                         and Customization guide.
                     "FilterId": The ID of the <filter/> element that is used to
                         initially populate the event list when the view is
                         selected.
                     "ViewType": Describes how the view was created: "context" (for
                         contextual) or "templatex" (for views created with the
                         wizard, where x is a number starting at 0 (zero) indicating
                         sequential position of template in eventViewTemplates.xml
                         file).
                     “UserChangedBase”: Indicates if the user changed the filter
                         using the wizard, which becomes the “base” filter in the
                         event list.
                     “UserChangedFilter”: Indicates if the user changed the filter
                         using the Filter function on the event list.
                     “UserChangedSort”: Indicates if the user changed the sort
                         using the Sort function on the event list.
                     “UserChangedColumns”: Indicates if the user changed the
                         columns using the Column Selector on the event list.
                     "EventImage": Image displayed for view in the event view list.
                     "ViewSource": Indicates “User” or “System” view.
                     "SystemView": "true" if system view’ “false” if user view.
                     "systemViewDate": Date and time the sysyem view was created.
                     "systemViewDescription": "true" if user can modify system view
                          description.
                     "systemViewFilter": "true" if user can modify system view
                          filter.
                     "systemViewBaseFilter": "true" if user can modify system view
                          base filter.
                     "systemViewSort": "true" if user can modify system view
                          sort.
                     "systemViewColumns": "true" if user can modify system view
                          columns.
                     "systemViewEventList": "true" if user can modify the
                         events included in the system view
                     "systemViewReplace": Indicates if the system view definition
                          will replace existing view of same name.
                     "systemViewStartDate": Date system view takes effect.
                     "systemViewEndDate": Date system view is no longer in effect.
                     "systemViewOwner": GUID of user that created system view.
                     "systemViewOwnerName": Name of user that created system view.
                     "systemViewRecipientGroups": GUID of groups specified as
                          recipients of the system view.
                     "systemViewRecipientPositions": GUID of positions specified as
                          recipients of the system view.
                     "systemViewRecipientResources": GUID of resources specified as
                          recipients of the system view.
                     "systemViewAuthorGroups": GUID of groups specified as
                          authors of the system view.
                     "systemViewAuthorPositions": GUID of positions specified as
                          authors of the system view.
                     "systemViewAuthorResources": GUID of resources specified as
                          authors of the system view.
                     "systemViewRecipientVersion": The major version number of the
                          organization model in which the recipient resides.
                     "systemViewAuthorVersion": The major version number of the
                          organization model in which the author resides.
                     "Filter": String value used to query the event collector
                         database for events that are to be displayed in the event
                         view.
                     "EventFilters": A <filters/> element, and one or more
```

```
                         subordinate <filter/> elements that specify the
                         pre-defined filters available in the event view.
                   }
                  ]
```

The `systemView...` items are included in the payload only if `SystemView=“true”`.

Example:

```
topic: com.tibco.wcc.order.wccPrototype.wccEventViews.listItemSelect
message: schemaId = com.tibco.wcc.schema.eventViews
         scope = "public"
         items = [
                   {
                     "View": "My Instances Started",
                     "Description": "Process instance started by me",
                     "DescriptionType": "summary",
                     "FilterId": "All",
                     "ViewType": "template3",
                     “UserChangedBase”: “true”
                     “UserChangedFilter”: “false”
                     “UserChangedSort”: “false”
                     “UserChangedColumns”: “false”
                     "EventImage": "JSXAPPS/base/application/images/
                        ProcessInstances.gif",
                     "ViewSource": “User”
                     "SystemView": “true”
                     "systemViewDate": "Tue Oct 2 09:22:07 PDT 2012",
                     "systemViewDescription": "true",
                     "systemViewFilter": "true",
                     "systemViewBaseFilter": "true",
                     "systemViewSort": "true",
                     "systemViewColumns": "true",
                     "systemViewEventList": "false",
                     "systemViewReplace": "true",
                     "systemViewStartDate": "Mon, 8 Oct 2012 07:00:00 UTC",
                     "systemViewEndDate": "Fri, 30 Nov 2012 08:00:00 UTC",
                     "systemViewOwner": "5395A979-AC8C-4D08-AD2F-F2790B449560",
                     "systemViewOwnerName": "Clint Hill",
                     "systemViewRecipientGroups": "",
                     "systemViewRecipientPositions": "",
                     "systemViewRecipientResources":
                         "07346908-E92E-4C1F-BF79-8EB6069EDECF",
                     "systemViewAuthorGroups": "",
                     "systemViewAuthorPositions": "",
                     "systemViewAuthorResources":
                         "A19F1BFB-0739-449A-8572-96EB4743E29D",
                     "systemViewRecipientVersion": "3",
                     "systemViewAuthorVersion": "3",
                     "Filter": "messageId='BX_INSTANCE_PROCESS_STARTED'",
                     "EventFilters": "<filters>\r\n <filter id=\"All\"
                        query=\"messageId='BX_INSTANCE_PROCESS_STARTED'\"
                        description=\"Process instance started events\"/>
                        \r\n </filters>",
                   }
                  ]
```

com.tibco.wcc.schema.eventViewsRemoved

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.eventViewsRemoved**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.eventViewsRemoved"
         scope = "public"
         items = [
                   {
                     "View": The event view being removed.
                     "Description": Description of the event view.
```

```
                    "DescriptionType": "detail" or "summary". These are defined in
                        the appropriate eventDescriptions.xml file. See the
                        Configuring Events chapter in the Workspace Configuration
                        and Customization guide.
                    "FilterId": The ID of the <filter/> element that is used to
                        initially populate the event list when the view is
                        selected.
                    "ViewType": Describes how the view was created: "context" (for
                        contextual) or "templatex" (for views created with the
                        wizard, where x is a number starting at 0 (zero) indicating
                        sequential position of template in eventViewTemplates.xml
                        file).
                    "UserChangedBase": Indicates if the user changed the filter
                        using the wizard, which becomes the "base" filter in the
                        event list.
                    "UserChangedFilter": Indicates if the user changed the filter
                        using the Filter function on the event list.
                    "UserChangedSort": Indicates if the user changed the sort
                        using the Sort function on the event list.
                    "UserChangedColumns": Indicates if the user changed the
                        columns using the Column Selector on the event list.
                    "EventImage": Image displayed for view in the event view list.
                    "ViewSource": Indicates "User" or "System" view.
                    "SystemView": "true" if system view' "false" if user view.
                    "systemViewDate": Date and time the sysyem view was created.
                    "systemViewDescription": "true" if user can modify system view
                         description.
                    "systemViewFilter": "true" if user can modify system view
                         filter.
                    "systemViewBaseFilter": "true" if user can modify system view
                         base filter.
                    "systemViewSort": "true" if user can modify system view
                         sort.
                    "systemViewColumns": "true" if user can modify system view
                         columns.
                    "systemViewEventList": "true" if user can modify the
                        events included in the system view
                    "systemViewReplace": Indicates if the system view definition
                         will replace existing view of same name.
                    "systemViewStartDate": Date system view takes effect.
                    "systemViewEndDate": Date system view is no longer in effect.
                    "systemViewOwner": GUID of user that created system view.
                    "systemViewOwnerName": Name of user that created system view.
                    "systemViewRecipientGroups": GUID of groups specified as
                         recipients of the system view.
                    "systemViewRecipientPositions": GUID of positions specified as
                         recipients of the system view.
                    "systemViewRecipientResources": GUID of resources specified as
                         recipients of the system view.
                    "systemViewAuthorGroups": GUID of groups specified as
                         authors of the system view.
                    "systemViewAuthorPositions": GUID of positions specified as
                         authors of the system view.
                    "systemViewAuthorResources": GUID of resources specified as
                         authors of the system view.
                    "systemViewRecipientVersion": The major version number of the
                         organization model in which the recipient resides.
                    "systemViewAuthorVersion": The major version number of the
                         organization model in which the author resides.
                    "Filter": String value used to query the event collector
                        database for events that are to be displayed in the event
                        view.
                  }
                ]
        remainingCount = The number of event views remaining after the removal.
```

The `systemView...` items are included in the payload only if `SystemView="true"`.

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccEventViews.listItemRemoved"
message: schemaId = "com.tibco.wcc.schema.eventViewsRemoved"
```

```
         scope = "public"
         items = [
                   {
                     "View": "My Instances Started",
                     "Description": "Process instances started by me",
                     "DescriptionType": "summary",
                     "FilterId": "All",
                     "ViewType": "template3",
                     “UserChangedBase”: “true”
                     “UserChangedFilter”: “false”
                     “UserChangedSort”: “false”
                     “UserChangedColumns”: “false”
                     "EventImage": "JSXAPPS/base/application/images/
                        ProcessInstances.gif",
                     "ViewSource": “User”
                     "SystemView": “true”
                     "systemViewDate": "Tue Oct 2 09:22:07 PDT 2012",
                     "systemViewDescription": "true",
                     "systemViewFilter": "true",
                     "systemViewBaseFilter": "true",
                     "systemViewSort": "true",
                     "systemViewColumns": "true",
                     "systemViewEventList": "false",
                     "systemViewReplace": "true",
                     "systemViewStartDate": "Mon, 8 Oct 2012 07:00:00 UTC",
                     "systemViewEndDate": "Fri, 30 Nov 2012 08:00:00 UTC",
                     "systemViewOwner": "5395A979-AC8C-4D08-AD2F-F2790B449560",
                     "systemViewOwnerName": "Clint Hill",
                     "systemViewRecipientGroups": "",
                     "systemViewRecipientPositions": "",
                     "systemViewRecipientResources":
                         "07346908-E92E-4C1F-BF79-8EB6069EDECF",
                     "systemViewAuthorGroups": "",
                     "systemViewAuthorPositions": "",
                     "systemViewAuthorResources":
                         "A19F1BFB-0739-449A-8572-96EB4743E29D",
                     "systemViewRecipientVersion": "3",
                     "systemViewAuthorVersion": "3",
                     "Filter": "messageId='BX_INSTANCE_PROCESS_STARTED'",
                     "EventFilters": "<filters>\r\n <filter id=\"All\"
                        query=\"messageId='BX_INSTANCE_PROCESS_STARTED'\"
                        description=\"Process instance started events\"/>
                        \r\n </filters>",
                   }
                 ]
         remainingCount = 6
```

com.tibco.wcc.schema.loginComplete

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.loginComplete**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.loginComplete"
         scope = "public"
         user = Name of the user who logged in.
         userGuid = GUID representing the user who logged in.
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccLogin.loginComplete"
message: schemaId = "com.tibco.wcc.schema.loginComplete"
         scope = "public"
         user = "Clint Hill"
         userGuid = "67140375-64B4-4639-B8C9-CB95AFC59E5A"
```

com.tibco.wcc.schema.organizationbrowser

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.organizationbrowser**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.organizationbrowser"
         scope = "public"
         modelVersion = The major version number of the organization model selected.
         overviewType = The type of entity selected: "POSITION", "GROUP",
            "ORGANIZATIONAL_UNIT", "ORGANIZATION", "CONTAINER", "HEADER" (if the
            Groups, Organizations, or LDAPContainers 'folder' is selected), or
            "HEADER_RESOURCES" (if the 'Resources' header is selected).
         overviewGuid = The GUID of the entity selected.
```

Example:

```
topic: "com.tibco.wcc.order.wccOrganizationBrowserWindowPrototype.
                                   wccOrganizationBrowser.organizationItemSelect"
message: schemaId = "com.tibco.wcc.schema.organizationbrowser"
         scope = "public"
         modelVersion = "3"
         overviewType = "POSITION"
         overviewGuid = "_0OgkOMpREd64gM7QE8RwxA"
```

com.tibco.wcc.schema.organizationresourcelist

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.organizationresourcelist**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.organizationresourcelist"
         scope = "public"
         items = [
                   {
                     "resourceGuid": The GUID of the resource selected.
                     "resourceName": The name of the resource selected.
                     "containerId": The ID of the container in which the resource
                           was created.
                     "containerName": The name of the container in which the
                           resource was created.
                   }
                 ]
```

Example:

```
topic: "com.tibco.wcc.Gerneric.wccOrganizationBrowserWindowPrototype.
                                   wccOrganizationResourceList.resourceItemSelect"
message: schemaId = "com.tibco.wcc.schema.organizationresourcelist"
         scope = "public"
         items = [
                   {
                     "resourceGuid": "67140375-64B4-4639-B8C9-CB95AFC59E5A",
                     "resourceName": "Clint Hill",
                     "containerId": "1",
                     "containerName": "local"
                   }
                 ]
```

com.tibco.wcc.schema.processInstances

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processInstances**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: INSTANCE.STATUS : The status of the instance: Active, Canceling,
            Completing, Failing, Halted, Halting, Not started, Restarting,
            Resuming, Starting, Suspended, or Suspending.
         INSTANCE.VERSION : The version of the process instance.
         INSTANCE.NAME : The name of the process instance.
         INSTANCE.ID : An identifier for the process instance.
         MODULE.NAME : The path to the XPDL file that defines the “process package”.
         INSTANCE.WAITING_WORK_COUNT : The number of outstanding work items in the
            process instance.
         INSTANCE.PRIORITY : The priority value of the process instance.
         INSTANCE.PARENT_PROCESS_ID : The instance ID of the parent process
            instance – only applicable for re-usable sub-processes.
         INSTANCE.START_DATE : The date and time the process instance was started.
         INSTANCE.ACTIVITY_FAULT_DATA : [Only applicable to halted process
            instances] The data that caused the instance to halt. For example,
            “com.tibco.bx.core.faults.BxException: Set Value “Result” failed:
            value=”?”; value class=java.lang.Double”.
         INSTANCE.ACTIVITY_FAULT_NAME : [Only applicable to halted process
            instances] The name of the fault. For example,
            “com.tibco.pvm.api.exceptions.PmModelException”.
         INSTANCE.FAILED_ACTIVITY_NAME : [Only applicable to halted process
            instances] The activity that caused the instance to halt. For example,
            “PerformDivision”.
         <custom field 1> = value  Custom fields and their values (to appear in the
                 .                 payload, the custom fields must appear in the
                 .                 columns of the process instance list).
                 .
         <custom field n> = value
         schemaId = "com.tibco.wcc.schema.processInstances"
         scope = "public"
         viewName = Name of the process view the instance is in.
         templateIds = The process templates included in the process view.
         haltedOnly = "true" if halted instance view; “false” if standard view.
         ids = [
                   Process instance ID.
               ]
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessInstances.listItemSelect"
message: INSTANCE.STATUS = "Active"
         INSTANCE.VERSION = "1.0.0.201305161548"
         INSTANCE.NAME = "HelpDeskProcess"
         INSTANCE.ID = "pvm:0a125"
         MODULE.NAME = "/HelpDesk/Process Packages/HelpDesk.xpdl"
         INSTANCE.WAITING_WORK_COUNT = "1"
         INSTANCE.PRIORITY = "200"
         INSTANCE.PARENT_PROCESS_ID = ""
         INSTANCE.START_DATE = "2013-05-28 14:00:31"
         INSTANCE.ACTIVITY_FAULT_DATA = ""
         INSTANCE.ACTIVITY_FAULT_NAME = ""
         INSTANCE.FAILED_ACTIVITY_NAME = ""
         ContactName = "Bonnie Brown"
         ContactPhone = "669-399-9090"
         IssueDescription = "Lost password"
         IssueDetails = ""
         ResolutionDescription = ""
         ResolutionDetails = ""
         schemaId = "com.tibco.wcc.schema.processInstances"
         scope = "public"
         viewName = "All Instances"
         templateIds = ""
         haltedOnly = "false"
         ids = [
                   "pvm:0a125"
               ]
```

com.tibco.wcc.schema.processInstancesShowOutstanding

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processInstancesShowOutstanding**.

The properties in the event message for this schema depends on whether the **Show Outstanding Work Items** or the **Show Supervised List of Outstanding Work Items** function is being invoked.

Show Outstanding Work Items

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.processInstancesShowOutstanding"
         scope = "public"
         ids = [
                  Process instance ID.
               ]
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessInstances.showOutstanding"
message: schemaId = "com.tibco.wcc.schema.processInstancesShowOutstanding"
         scope = "public"
         ids = [
                  "pvm:0a121"
               ]
```

Show Supervised List of Outstanding Work Items

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.processInstancesShowOutstanding"
         scope = "public"
         adminObjs = [
                        {
                          "entityType": The type of entity selected: "POSITION",
                              "GROUP","ORGANIZATIONAL_UNIT", or "RESOURCE".
                          "guid": The GUID of the entity selected.
                          "displayName": The entity selected.
                        }
                      ]
         ids = [
                 Process instance ID whose outstanding work items are being
                      displayed.
               ]
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessInstances.showOutstanding"
message: schemaId = "com.tibco.wcc.schema.processInstancesShowOutstanding"
         scope = "public"
         adminObjs = [
                        {
                          "entityType": "POSITION",
                          "guid": "_2ieLgMpREd64gM7QE8RwxA",
                          "displayName": "CustomerServiceRepresentative"
                        }
                      ]
         ids = [
                 "pvm:0a123"
               ]
```

com.tibco.wcc.schema.processtemplates

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processtemplates**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
         scope = "public"
         listIds = Array of process templates selected in the list. Includes the
                         version of the process templates.
```

Example:

```
topic: "com.tibco.wcc.Generic.wccPrototype.wccProcessTemplates.listItemSelect"
message: schemaId = "com.tibco.wcc.schema.processtemplates"
         scope = "public"
         listIds = [
                     "SalesCallbackProcess~/SalesCallback/Process Packages/
                         SalesCallback.xpdl~1.0.0.201005251718",
                     "SalesSurveyProcess~/SalesSurvey/Process Packages/
                         SalesSurvey.xpdl~1.0.1.201005251719"
                   ]
```

com.tibco.wcc.schema.processtemplatesex

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processtemplatesex**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
         scope = "public"
         checkedIds = Array of process templates checked in the list. Includes the
                         version of the process templates.
```

Example:

```
topic: "com.tibco.wcc.Generic.wccPrototype.wccProcessTemplatesEx.listItemSelect"
message: schemaId = "com.tibco.wcc.schema.processtemplatesex"
         scope = "public"
         checkedIds = [
                        "SalesCallbackProcess~*~*",
                        "SalesCallbackProcess~/SalesCallback/Process
                             Packages/SalesCallback.xpdl~1.0.0.201005251718",
                        "SalesSurveyProcess~*~*",
                        "SalesSurveyProcess~/SalesSurvey/Process
                             Packages/SalesSurvey.xpdl~1.0.1.201005251719"
                      ]
```

com.tibco.wcc.schema.processtemplatesexrefresh

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processtemplatesexrefresh**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
         scope = "public"
         checkedIds = Array of process templates checked in the list(1). Includes
the
                         version of the process templates.
```

Example:

```
topic: "com.tibco.wcc.Generic.wccPrototype.wccProcessTemplatesEx.listRefresh"
message: schemaId = "com.tibco.wcc.schema.processtemplatesexrefresh"
         scope = "public"
         checkedIds = [
                        "SalesCallbackProcess~*~*",
                        "SalesCallbackProcess~/SalesCallback/Process
                             Packages/SalesCallback.xpdl~1.0.0.201005251718",
                        "SalesSurveyProcess~*~*",
                        "SalesSurveyProcess~/SalesSurvey/Process
                             Packages/SalesSurvey.xpdl~1.0.1.201005251719"
                      ]
```

[7] All templates/versions that are selected (checked) when a refresh is performed are included in the event message, although they are automatically unchecked in the UI after the refresh.

com.tibco.wcc.schema.processViews

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processViews**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.processViews"
         scope = "public"
         items = [
                   {
                     "View": Name of the process view.
                     "Description": Description of the process view.
                     "Type": Type of process view:"UserEdit" (for views created
                         using the wizard) or "Default" (for the default All
                         Instances view).
                     "TemplateIds": Indicates the processes included in the process
                         view, each separated by ~*~*, characters (this is simply
                         "*" for the default All Instances view).
                     "UserChangedBase": Indicates if the user changed the filter
                         using the wizard, which becomes the "base" filter in the
                         process instance list.
                     "UserChangedFilter": Indicates if the user changed the filter
                         using the Filter function on the process instance list.
                     "UserChangedSort": Indicates if the user changed the sort
                         using the Sort function on the process instance list.
                     "UserChangedColumns": Indicates if the user changed the
                         columns using the Column Selector on the process instance
                         list.
                     "HaltedOnly": "true" if process view is for halted views;
                         "false" if it is for standard views.
                     "Image": Image displayed in the process view list.
                     "ViewSource": Indicates it is a "User" or "System" view.
                     "SystemView": "true" if system view' "false" if user view.
                     "systemViewDate": Date and time the sysyem view was created.
                     "systemViewDescription": "true" if user can modify system view
                          description.
                     "systemViewFilter": "true" if user can modify system view
                          filter.
                     "systemViewBaseFilter": "true" if user can modify system view
                          base filter.
                     "systemViewSort": "true" if user can modify system view
                          sort.
                     "systemViewColumns": "true" if user can modify system view
                          columns.
                     "systemViewProcessList": "true" if user can modify the
                         processes included in the system view
                     "systemViewReplace": Indicates if the system view definition
                          will replace existing view of same name.
                     "systemViewStartDate": Date system view takes effect.
                     "systemViewEndDate": Date system view is no longer in effect.
                     "systemViewOwner": GUID of user that created system view.
                     "systemViewOwnerName": Name of user that created system view.
                     "systemViewRecipientGroups": GUID of groups specified as
                          recipients of the system view.
                     "systemViewRecipientPositions": GUID of positions specified as
                          recipients of the system view.
                     "systemViewRecipientResources": GUID of resources specified as
                          recipients of the system view.
                     "systemViewAuthorGroups": GUID of groups specified as
                          authors of the system view.
                     "systemViewAuthorPositions": GUID of positions specified as
                          authors of the system view.
                     "systemViewAuthorResources": GUID of resources specified as
                          authors of the system view.
                     "systemViewRecipientVersion": The major version number of the
                          organization model in which the recipient resides.
                     "systemViewAuthorVersion": The major version number of the
                          organization model in which the author resides.
                     "Filter": The query string used to create the process view.
                   }
                 ]
```

The systemView... items are included in the payload only if SystemView=“true”.

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessViews.listItemSelect"
message: schemaId = "com.tibco.wcc.schema.processViews"
         scope = "public"
         items = [
                   {
                     "View": "Sales",
                     "Description": "Sales Leads",
                     "Type": "UserEdit",
                     "TemplateIds": "SalesCallbackProcess~*~*",
                     "UserChangedBase": "true",
                     "UserChangedFilter": "false",
                     "UserChangedSort": "false",
                     "UserChangedColumns": "false",
                     "HaltedOnly": "false",
                     "Image": "JSXAPPS/base/application/images/
ProcessInstances.gif",
                     "ViewSource": "System",
                     "SystemView": "true",
                     "systemViewDate": "Tue Oct 2 09:10:14 PDT 2012",
                     "systemViewDescription": "true",
                     "systemViewFilter": "true",
                     "systemViewBaseFilter": "true",
                     "systemViewSort": "true",
                     "systemViewColumns": "true",
                     "systemViewProcessList": "false",
                     "systemViewReplace": "true",
                     "systemViewStartDate": "Wed, 3 Oct 2012 07:00:00 UTC",
                     "systemViewEndDate": "Wed, 31 Oct 2012 07:00:00 UTC",
                     "systemViewOwner": "5395A979-AC8C-4D08-AD2F-F2790B449560",
                     "systemViewOwnerName": "Clint Hill",
                     "systemViewRecipientGroups": "",
                     "systemViewRecipientPositions": "_ZKnCcDBTEd-x4cVy01Xlww",
                     "systemViewRecipientResources": "",
                     "systemViewAuthorGroups": "",
                     "systemViewAuthorPositions": "_0Ogk0MpREd64gM7QE8RwxA",
                     "systemViewAuthorResources": "",
                     "systemViewRecipientVersion": "3",
                     "systemViewAuthorVersion": "3",
                     "Filter": " INSTANCE.PRIORITY = 1 "
                   }
                 ]
```

com.tibco.wcc.schema.processViewsRemoved

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.processViewsRemoved**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.processViewsRemoved"
         scope = "public"
         items = [
                   {
                     "View": Name of the process view being removed.
                     "Description": Description of the process view.
                     "Type": Type of process view:"UserEdit" (for views created
                         using the wizard) or "Default" (for the default All
                         Instances view).
                     "TemplateIds": The processes included in the process view,
                         each separated by ~*~*, characters (this is simply "*" for
                         the default All Instances view).
                     “UserChangedBase": Indicates if the user changed the filter
```

```
                                using the wizard, which becomes the “base” filter in the
                                process instance list.
                            “UserChangedFilter”: Indicates if the user changed the filter
                                using the Filter function on the process instance list.
                            “UserChangedSort”: Indicates if the user changed the sort
                                using the Sort function on the process instance list.
                            “UserChangedColumns”: Indicates if the user changed the
                                columns using the Column Selector on the process instance
                                list.
                            "HaltedOnly": “true” if process view is for halted views;
                                “false” if it is for standard views.
                            “Image”: Image displayed in the process view list.
                            “ViewSource”: Indicates it is a “User” or “System” view.
                            "SystemView": "true" if system view’ “false” if user view.
                            "systemViewDate": Date and time the sysyem view was created.
                            "systemViewDescription": "true" if user can modify system view
                                 description.
                            "systemViewFilter": "true" if user can modify system view
                                 filter.
                            "systemViewBaseFilter": "true" if user can modify system view
                                 base filter.
                            "systemViewSort": "true" if user can modify system view
                                 sort.
                            "systemViewColumns": "true" if user can modify system view
                                 columns.
                            "systemViewProcessList": "true" if user can modify the
                                processes included in the system view
                            "systemViewReplace": Indicates if the system view definition
                                 will replace existing view of same name.
                            "systemViewStartDate": Date system view takes effect.
                            "systemViewEndDate": Date system view is no longer in effect.
                            "systemViewOwner": GUID of user that created system view.
                            "systemViewOwnerName": Name of user that created system view.
                            "systemViewRecipientGroups": GUID of groups specified as
                                 recipients of the system view.
                            "systemViewRecipientPositions": GUID of positions specified as
                                 recipients of the system view.
                            "systemViewRecipientResources": GUID of resources specified as
                                 recipients of the system view.
                            "systemViewAuthorGroups": GUID of groups specified as
                                 authors of the system view.
                            "systemViewAuthorPositions": GUID of positions specified as
                                 authors of the system view.
                            "systemViewAuthorResources": GUID of resources specified as
                                 authors of the system view.
                            "systemViewRecipientVersion": The major version number of the
                                 organization model in which the recipient resides.
                            "systemViewAuthorVersion": The major version number of the
                                 organization model in which the author resides.
                            "systemViewOrgVersion": Version of organization model.
                            "Filter": The query string used to create the process view.
                        }
                      ]
         remainingCount = Number of process views remaining after the removal.
```

The `systemView...` items are included in the payload only if `SystemView=“true”`.

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessViews.listItemRemoved"
message: schemaId = "com.tibco.wcc.schema.processViewsRemoved"
         scope = "public"
         items = [
                   {
                     "View": "Sales",
                     "Description": "Sales Leads",
                     "Type": "UserEdit",
                     "TemplateIds": "SalesCallbackProcess~*~*",
                     "UserChangedBase": "true",
                     "UserChangedFilter": "false",
                     "UserChangedSort": "false",
                     "UserChangedColumns": "false",
                     "HaltedOnly": "false",
                     "Image": "JSXAPPS/base/application/images/
ProcessInstances.gif",
                     "ViewSource": "System",
                     "SystemView": "true",
                     "systemViewDate": "Tue Oct 2 09:10:14 PDT 2012",
                     "systemViewDescription": "true",
                     "systemViewFilter": "true",
                     "systemViewBaseFilter": "true",
                     "systemViewSort": "true",
                     "systemViewColumns": "true",
                     "systemViewProcessList": "false",
                     "systemViewReplace": "true",
                     "systemViewStartDate": "Wed, 3 Oct 2012 07:00:00 UTC",
                     "systemViewEndDate": "Wed, 31 Oct 2012 07:00:00 UTC",
                     "systemViewOwner": "5395A979-AC8C-4D08-AD2F-F2790B449560",
                     "systemViewOwnerName": "Clint Hill",
                     "systemViewRecipientGroups": "",
                     "systemViewRecipientPositions": "_ZKnCcDBTEd-x4cVy01Xlww",
                     "systemViewRecipientResources": "",
                     "systemViewAuthorGroups": "",
                     "systemViewAuthorPositions": "_0Ogk0MpREd64gM7QE8RwxA",
                     "systemViewAuthorResources": "",
                     "systemViewRecipientVersion": "3",
                     "systemViewAuthorVersion": "3",
                     "Filter": " INSTANCE.PRIORITY = 1 "
                   }
                 ]
         remainingCount = 2
```

com.tibco.wcc.schema.saveEventView

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.saveEventView**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.saveEventView"
         scope = "private"
```

Example:

```
N/A - This event is private.
```

com.tibco.wcc.schema.saveEventViewAs

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.saveEventViewAs**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.saveEventViewAs"
         scope = "private"
```

Example:

```
N/A - This event is private.
```

com.tibco.wcc.schema.saveProcessView

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.saveProcessView**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.saveProcessView"
         scope = "private"
```

Example:

```
N/A - This event is private.
```

com.tibco.wcc.schema.saveProcessViewAs

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.saveProcessViewAs**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.saveProcessViewAs"
         scope = "private"
```

Example:

```
N/A - This event is private.
```

com.tibco.wcc.schema.saveWorkView

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.saveWorkView**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.saveWorkView"
         scope = "private"
```

Example:

```
N/A - This event is private.
```

com.tibco.wcc.schema.saveWorkViewAs

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.saveWorkViewAs**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.saveWorkViewAs"
         scope = "private"
```

Example:

```
N/A - This event is private.
```

com.tibco.wcc.schema.showEvents

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.showEvents**.

The event message for this schema comes in two formats, denoted by a version number in the **schemaVersion** property:

- **schemaVersion=1**: This defines the format of the data passed in the **showEvents** event message when the event is published by a WCC component (Process Instances, Work Items, Organization Browser, and Organization Resource List). This event format can be captured and used as needed.
- **schemaVersion=2**: This defines the format of the **showEvents** PageBus event that can be published by an external application. This version of the **showEvents** event message format is handled by the WCC **Event Views** component to create a temporary event view and display it in a WCC **Event Viewer** component.

Data in the PageBus event — **schemaVersion=1**:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
topic: "com.tibco.wcc.order.wccPrototype.wccWorkItems.showEvents"
message: schemaId = "com.tibco.wcc.schema.showEvents"
         scope = "public"
         schemaVersion = Indicates whether the event is being published by a WCC
             component or by an external application -- see description above.
         view = Name of the event view being created.
         descriptionType = "detail" or "summary". These are defined in the
```

```
                 appropriate eventDescriptions.xml file. See the Configuring Events
                 chapter in the Workspace Configuration and Customization guide.
            filterId = The ID of the <filter/> element that is used to initially
                 populate the event list when the view is selected.
            eventImage = Image displayed for view in the event view list.
            eventNode = Defines the item (work item, process instance, etc.) selected
                 when the show events function is initiated.
            eventLinkNode = Defines the event view created by the selected show events
                 drop-down menu selected. See associated <link/> record in
                 eventLinks.xml.
            viewType = Always "context" to denote the new event view is in the context
                 of the item (work item, process instance, etc.) selected when the show
                 events function is initiated.
            ids = (Only appears in message for show events from the process instance
                 list.) An array containing the ID of the process instance selected when
                 the show events function is initiated.
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessInstances.showEvents"
message: schemaId = "com.tibco.wcc.schema.showEvents"
         scope = "public"
         schemaVersion = "1"
         view = "SalesCallbackProcess instances"
         descriptionType = "summary"
         filterId = "All"
         eventImage = "JSXAPPS/base/application/images/ProcessTemplate.gif"
         eventNode = "<record jsxid="id0x0937e1a0" jsxtext="" jsximg=""
             StatusImage="JSXAPPS/base/application/images/
ProcessInstanceActive.gif             " INSTANCE.VERSION="1.0.0.201005251718"
             INSTANCE.NAME="SalesCallbackProcess"
                .
                .
                .
         eventLinkNode = "<record jsximg="JSXAPPS/base/application/images/
             ProcessTemplate.gif" descriptionType="summary" defaultFilterId="All"
             jsxtext="Instances of this process" jsxexecute="this.getAncestorOfType
             ('com.tibco.wcc.base.ListContainer').showEventViewer(this.getRecordNod
            e(this.getValue()))" name="{INSTANCE.NAME} instances"
                .
                .
                .
         viewType = "context"
         ids = [
                  "pvm:0a123"
               ]
```

Data in the PageBus event — **schemaVersion=2**:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
topic: "com.tibco.wcc.order.wccPrototype.wccWorkItems.showEvents"
message: schemaId = "com.tibco.wcc.schema.showEvents"
         scope = "public"
         schemaVersion = Indicates whether the event is being published by a WCC
             component or by an external application -- see description above.
         eventLinkId = The <links/> element messageId, which defines the contextual
             or event link to add. These are defined in eventLink.xml. For example:
             <links messageId="WorkItem_WorkspaceContext">.
         substitutionDataXml = The XML containing the attributes that are
             substituted into the <link/> element that is used. The root element
             name is record: <record att1="val1" att2="val2"/>
         menuText = If there are multiple <link/> elements, the menuText value is
             used to locate a specific one. If null, the first link is used.
         eventLinkXml = The XML that defines the event links. If this is null, the
             application-configured eventLinks XML is used.
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccProcessInstances.showEvents"
message: schemaId = "com.tibco.wcc.schema.showEvents"
         scope = "public"
         schemaVersion = "2"
         eventLinkId = "WorkItem_WorkspaceContext"
         substitutionDataXml = "<record ActivityID="pvm:001i8"
```

```
             AppInstance="pvm:0a121" AppName="CSCallbackProcess" ID="2"
             Name="ManagerReview"/>"
         menuText = "My activity for this work item"
         eventLinkXml = ""
```

com.tibco.wcc.schema.startinstance

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.startinstance**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.startInstance"
         scope = "public"
         name = The name of the process template of which an instance is being
            started.
         module = The path to the XPDL file that defines the "process package".
         version = The version of the process template from which the instance was
            started.
```

Example:

```
topic: "com.tibco.wcc.Test1.wccPrototype.wccStartInstance.instanceStarted"
message: schemaId = "com.tibco.wcc.schema.startInstance"
         scope = "public"
         name = "InternalHelpDesk"
         module = "/HelpDesk/Process Packages/HelpDesk.xpdl"
         version = "1.0.0.201012081627"
```

com.tibco.wcc.schema.workItems

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.workItems**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.workItems"
         scope = "public"
         id = Work item ID.
         version = Number of times the work item has changed state. Starts at 0 when
            the work item is created, and is incremented by one each time it changes
            state.
         description = Work item description.
         startDate = Date and time the work item was created and arrived in
            the work item list.
         endDate = Date and time in this column indicates that the work item has a
            deadline. Note that by default, the column header is “Target Date”.
         distributionStrategy = Method used to distribute the work item when it was
            originally created, either "OFFER" or "ALLOCATE".
         priority = Numeric value indicating the relative importance of the
            work item.
         groupId = Identifies a number of work items that are grouped together in
            the process definition for some purpose.
         activityId = Identifies the user activity within the process that generated
            the work item. This number can be used in the event viewer to identify
            the work item across all components.
         appId = A GUID that represents the application.
         appInstance = Id of the process instance from which the work item was
            created.
         appName = Name of process template that was started to create this work
            item.
         scheduleStatus = Indicates if the work item is inside its schedule period,
            which is considered from the start date/time (Start Date column) to the
            target date/time (Target Date column). The possible values are: BEFORE,
            DURING, AFTER and NO_SCHEDULE.
         scheduleStatusLocalized = The localized term for “status”.
         state = The work item’s current state: OFFERED, ALLOCATED, CREATED, OPENED,
            PENDED, PENDHIDDEN, or SUSPENDED.
         stateLocalized = The localized term for “state”.
         workTypeId = Work type ID.
         workTypeUid = Work type universal ID.
         workTypeVersion = Work type version number.
         workTypeDescription = Work type description.
         attribute1 = The value in attribute 1.
         attribute2 = The value in attribute 2.
         attribute3 = The value in attribute 3.
         attribute4 = The value in attribute 4.
         attribute5 = The value in attribute 5.
         attribute6 = The value in attribute 6.
         attribute7 = The value in attribute 7.
         attribute8 = The value in attribute 8.
         attribute9 = The value in attribute 9.
         attribute10 = The value in attribute 10.
         attribute11 = The value in attribute 11.
         attribute12 = The value in attribute 12.
         attribute13 = The value in attribute 13.
         attribute14 = The value in attribute 14.
         attribute15 = The value in attribute 15.
         attribute16 = The value in attribute 16.
         attribute17 = The value in attribute 17.
         attribute18 = The value in attribute 18.
         attribute19 = The value in attribute 19.
         attribute20 = The value in attribute 20.
         attribute21 = The value in attribute 21.
         attribute22 = The value in attribute 22.
         attribute23 = The value in attribute 23.
         attribute24 = The value in attribute 24.
         attribute25 = The value in attribute 25.
         attribute26 = The value in attribute 26.
         attribute27 = The value in attribute 27.
         attribute28 = The value in attribute 28.
         attribute29 = The value in attribute 29.
         attribute30 = The value in attribute 30.
         attribute31 = The value in attribute 31.
         attribute32 = The value in attribute 32.
         attribute33 = The value in attribute 33.
         attribute34 = The value in attribute 34.
         attribute35 = The value in attribute 35.
         attribute36 = The value in attribute 36.
         attribute37 = The value in attribute 37.
         attribute38 = The value in attribute 38.
         attribute39 = The value in attribute 39.
         attribute40 = The value in attribute 40.
```

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccWorkItems.listItemSelect"
message: schemaId = "com.tibco.wcc.schema.workItems"
         scope = "public"
         id = "52"
         version = "0"
         description = "Return Call"
         startDate = "2010-10-18 10:28:49"
         endDate = ""
         distributionStrategy = "OFFER"
         priority = "50"
         groupId = "0"
         activityId = "pvm:001ij"
         appId = "_uxr1YF6yEd-KqJmYNEw8ow"
         appInstance = "pvm:0a123"
         appName = "SalesCallbackProcess"
         scheduleStatus = "BEFORE"
         scheduleStatusLocalized = "Before"
         state = "OFFERED"
         stateLocalized = “Offered”
         workTypeId = ""
         workTypeUid = ""
         workTypeVersion = ""
         workTypeDescription = ""
         attribute1 = ""
         attribute2 = ""
         attribute3 = ""
         attribute4 = ""
         attribute5 = ""
         attribute6 = ""
         attribute7 = ""
         attribute8 = ""
         attribute9 = ""
         attribute10 = ""
         attribute11 = ""
         attribute12 = ""
         attribute13 = ""
         attribute14 = ""
```

com.tibco.wcc.schema.openNextWorkItem

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.openNextWorkItem**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.openNextWorkItem"
         scope = "public"
```

Note - This event message has no payload.

Example:

```
topic: "com.tibco.wcc.Gerneric.wccPrototype.wccWorkItems.openNextItem"
       message: schemaId = "com.tibco.wcc.schema.openNextWorkItem"
                scope = "public"
```

com.tibco.wcc.schema.workviews

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.workviews**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.workviews"
         scope = "public"
         items = [
                   {
                     "View": Name of the work view.
                     "Description": Description of the work view.
                     “UserChangedBase”: Indicates if the user changed the filter
                         using the wizard, which becomes the “base” filter in the
                         work item list.
                     “UserChangedFilter”: Indicates if the user changed the filter
                         using the Filter function on the work item list.
                     “UserChangedSort”: Indicates if the user changed the sort
                         using the Sort function on the work item list.
                     “UserChangedColumns”: Indicates if the user changed the
                         columns using the Column Selector on the work item
                         list.
                     “ViewSource”: Indicates it is a “User” or “System” view.
                     "SystemView": "true" if system view’ “false” if user view.
                     "systemViewDate": Date and time the sysyem view was created.
                     "systemViewDescription": "true" if user can modify system view
                          description.
                     "systemViewFilter": "true" if user can modify system view
                          filter.
                     "systemViewBaseFilter": "true" if user can modify system view
                          base filter.
                     "systemViewSort": "true" if user can modify system view
                          sort.
                     "systemViewColumns": "true" if user can modify system view
                          columns.
                     "systemViewWorkList": "true" if user can select
                          groups/positions/resources for supervised work list.
                     "systemViewReplace": Indicates if the system view definition
                          will replace existing view of same name.
                     "systemViewStartDate": Date system view takes effect.
                     "systemViewEndDate": Date system view is no longer in effect.
                     "systemViewOwner": GUID of user that created system view.
                     "systemViewOwnerName": Name of user that created system view.
                     "systemViewRecipientGroups": GUID of groups specified as
                          recipients of the system view.
                     "systemViewRecipientPositions": GUID of positions specified as
                          recipients of the system view.
                     "systemViewRecipientResources": GUID of resources specified as
                          recipients of the system view.
                     "systemViewAuthorGroups": GUID of groups specified as
                          authors of the system view.
                     "systemViewAuthorPositions": GUID of positions specified as
                          authors of the system view.
                     "systemViewAuthorResources": GUID of resources specified as
                          authors of the system view.
                     "systemViewRecipientVersion": The major version number of the
                          organization model in which the recipient resides.
                     "systemViewAuthorVersion": The major version number of the
                          organization model in which the author resides.
                     "systemViewOrgVersion": Version of organization model.
                     "Type": Type of work view:"UserEdit" (for views created
                         using the wizard, as well as views created with one of the
                         'Show Outstanding' functions on the process instance list)
                          or "Default" (for the default Inbox).
                     "Filter": The filter string for the view.
                     "ParentNode": The node under which the work view was created:
                         either "MYWORK" or "SUPERVISEDWORK".
                     "entityType": (Only appears for Supervised work views.) The
                         type of entity being supervised: "RESOURCE", "GROUP",
                         "ORGANIZATIONAL_UNIT", or "POSITION".
                     "guid": (Only appears for Supervised work views.) The GUID
                          representing the entity being supervised.
                     "modelVersion": (Only appears for Supervised work views.) The
                          version of the organization model.
                     "SupervisedWorkItemState": (Only appears for Supervised work
                          views.) Indicates whether the work items were offered or
                          allocated to the organizational entity.
                     "temporary": (Only appears when the work view is created with
                          one of the 'Show Outstanding' functions on the process
                          instance list) A value of 'true" indicates the work view
                          is still temporary.
                   }
                 ]
```

The `systemView...` items are included in the payload only if `SystemView="true"`.

Example:

```
topic: "com.tibco.wcc.order.wccPrototype.wccWorkViews.listItemSelect"
message: schemaId = "com.tibco.wcc.schema.workviews"
         scope = "public"
         items = [
                   {
                       "View": "Urgent",
                       "Description": "Past due claims",
                       "UserChangedBase": "true",
                       "UserChangedFilter": "false",
                       "UserChangedSort": "false",
                       "UserChangedColumns": "false",
                       "ViewSource": "System",
                       "SystemView": "true",
                       "systemViewDate": "Tue Oct 2 10:27:54 PDT 2012",
                       "systemViewDescription": "true",
                       "systemViewFilter": "false",
                       "systemViewBaseFilter": "true",
                       "systemViewSort": "true",
                       "systemViewColumns": "true",
                       "systemViewReplace": "true",
                       "systemViewStartDate": "Mon, 1 Oct 2012 07:00:00 UTC",
                       "systemViewEndDate": "Wed, 31 Oct 2012 07:00:00 UTC",
                       "systemViewOwner": "5395A979-AC8C-4D08-AD2F-F2790B449560",
                       "systemViewOwnerName": "Clint Hill",
                       "systemViewRecipientGroups": "",
                       "systemViewRecipientPositions": "",
                       "systemViewRecipientResources":
                           "07346908-E92E-4C1F-BF79-8EB6069EDECF",
                       "systemViewAuthorGroups": "",
                       "systemViewAuthorPositions": "",
                       "systemViewAuthorResources":
                           "A19F1BFB-0739-449A-8572-96EB4743E29D",
                       "systemViewRecipientVersion": "3",
                       "systemViewAuthorVersion": "3",
                       "Type": "UserEdit",
                       "Filter": " endDate < {~TODAY~} ",
                       ParentNode": "MYWORK"
                   }
                 ]
```

com.tibco.wcc.schema.workviewsRemoved

This topic describes the data passed in the payload for PageBus event, **com.tibco.wcc.schema.workviewsRemoved**.

Data in the PageBus event:

```
topic: The PageBus topic for the event -- see Non-WCC Components Publishing Events.
message: schemaId = "com.tibco.wcc.schema.workviewsRemoved"
         scope = "public"
         items = [
                   {
                     "View": Name of the work view.
                     "Description": Description of the work view.
                     “UserChangedBase”: Indicates if the user changed the filter
                         using the wizard, which becomes the “base” filter in the
                         work item list.
                     “UserChangedFilter”: Indicates if the user changed the filter
                         using the Filter function on the work item list.
                     “UserChangedSort”: Indicates if the user changed the sort
                         using the Sort function on the work item list.
                     “UserChangedColumns”: Indicates if the user changed the
                         columns using the Column Selector on the work item
                         list.
                     “ViewSource”: Indicates it is a “User” or “System” view.
                     "SystemView": "true" if system view’ “false” if user view.
                     "systemViewDate": Date and time the sysyem view was created.
                     "systemViewDescription": "true" if user can modify system view
                          description.
                     "systemViewFilter": "true" if user can modify system view
                          filter.
                     "systemViewBaseFilter": "true" if user can modify system view
                          base filter.
                     "systemViewSort": "true" if user can modify system view
                          sort.
                     "systemViewColumns": "true" if user can modify system view
                          columns.
                     "systemViewReplace": Indicates if the system view definition
                          will replace existing view of same name.
                     "systemViewStartDate": Date system view takes effect.
                     "systemViewEndDate": Date system view is no longer in effect.
                     "systemViewOwner": GUID of user that created system view.
                     "systemViewOwnerName": Name of user that created system view.
                     "systemViewRecipientGroups": GUID of groups specified as
                          recipients of the system view.
                     "systemViewRecipientPositions": GUID of positions specified as
                          recipients of the system view.
                     "systemViewRecipientResources": GUID of resources specified as
                          recipients of the system view.
                     "systemViewAuthorGroups": GUID of groups specified as
                          authors of the system view.
                     "systemViewAuthorPositions": GUID of positions specified as
                          authors of the system view.
                     "systemViewAuthorResources": GUID of resources specified as
```

```
                             authors of the system view.
                        "systemViewRecipientVersion": The major version number of the
                             organization model in which the recipient resides.
                        "systemViewAuthorVersion": The major version number of the
                             organization model in which the author resides.
                        "systemViewOrgVersion": Version of organization model.
                        "Type": Type of work view:"UserEdit" (for views created
                            using the wizard, as well as views created with one of the
                            'Show Outstanding' functions on the process instance list)
                             or "Default" (for the default Inbox).
                        "Filter": The filter string for the view.
                        "ParentNode": The node under which the work view was created:
                            either "MYWORK" or "SUPERVISEDWORK".
                        "entityType": (Only appears for Supervised work views.) The
                            type of entity being supervised: "RESOURCE", "GROUP",
                            "ORGANIZATIONAL_UNIT", or "POSITION".
                        "guid": (Only appears for Supervised work views.) The GUID
                             representing the entity being supervised.
                        "modelVersion": (Only appears for Supervised work views.) The
                             version of the organization model.
                        "SupervisedWorkItemState": (Only appears for Supervised work
                             views.) Indicates whether the work items were offered or
                             allocated to the organizational entity.
                        "temporary": (Only appears when the work view is created with
                             one of the 'Show Outstanding' functions on the process
                             instance list) A value of 'true" indicates the work view
                             is still temporary.
                    }
                  ]
          remainingCount = The number of work views remaining after the removal.
```

The `systemView...` items are included in the payload only if `SystemView="true"`.

Example:

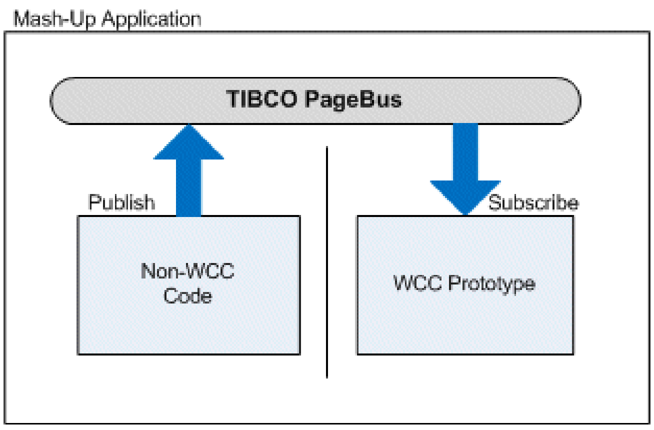
```
topic: "com.tibco.wcc.order.wccPrototype.wccWorkViews.listItemSelect"
message: schemaId = "com.tibco.wcc.schema.workviewsRemoved"
         scope = "public"
         items = [
                    {
                       "View": "Urgent",
                       "Description": "Past due claims",
                       "UserChangedBase": "true",
                       "UserChangedFilter": "false",
                       "UserChangedSort": "false",
                       "UserChangedColumns": "false",
                       "ViewSource": "System",
                       "SystemView": "true",
                       "systemViewDate": "Tue Oct 2 10:27:54 PDT 2012",
                       "systemViewDescription": "true",
                       "systemViewFilter": "false",
                       "systemViewBaseFilter": "true",
                       "systemViewSort": "true",
                       "systemViewColumns": "true",
                       "systemViewReplace": "true",
                       "systemViewStartDate": "Mon, 1 Oct 2012 07:00:00 UTC",
                       "systemViewEndDate": "Wed, 31 Oct 2012 07:00:00 UTC",
                       "systemViewOwner": "5395A979-AC8C-4D08-AD2F-F2790B449560",
                       "systemViewOwnerName": "Clint Hill",
                       "systemViewRecipientGroups": "",
                       "systemViewRecipientPositions": "",
                       "systemViewRecipientResources":
                           "07346908-E92E-4C1F-BF79-8EB6069EDECF",
                       "systemViewAuthorGroups": "",
                       "systemViewAuthorPositions": "",
                       "systemViewAuthorResources":
                           "A19F1BFB-0739-449A-8572-96EB4743E29D",
                       "systemViewRecipientVersion": "3",
                       "systemViewAuthorVersion": "3",
                       "Type": "UserEdit",
                       "Filter": " endDate < {~TODAY~} ",
                       ParentNode": "MYWORK"
                    }
                ]
            remainingCount = 2
```

Non-WCC Code Publishing Events

Non-WCC code can also publish an event to the PageBus.

This causes one of the following to be displayed in the mash-up application:

- work item list
- process instance list
- event list(1) (also called the Event Viewer)



This allows non-WCC code to display a specific work item list or process instance list, and control it from the non-WCC part of the mash-up application. (Note that if you want the user to be able to display custom lists (that is, work item or process instance *views*), rather than a specific work item or process instance list, then a better option would be to actually display the Work Views or Process Views component.)

The non-WCC code must use the **Pagebus.publish** method to publish the appropriate event, depending on whether you want a work item list or process instance list displayed. (A **publishPassThruEvent** method is provided in the **IframeContainer** class to publish the event if you are using the PageBus Managed Hub in an iframe-to-iframe configuration — see Using the TIBCO PageBus Managed Hub with WCC Components.)

To display the work item list or process instance list, the WCC client application must subscribe to the event that was published by the non-WCC code. This is done using TIBCO General Interface builder. See Subscribing to an Event Published by Non-WCC Code.

Two sample applications are provided that assist you in generating the event that represents the desired work item or process instance list, then publish that event so that the work item or process instance list is displayed:

- **pageBusPayloadHelperApp** - This application provides a wizard that guides you through building the desired work item or process instance list, then generates the PageBus event payload.

  For more information, see pageBusPayloadHelperApp.
- **pageBusPayloadPublishApp** - This application is used to publish the event message payload you generated with the pageBusPayloadHelperApp application. It renders the work item or process instance list that is described in the payload.

[8] The information here is specifically about displaying a work item or process instance list. An event list (the Event Viewer) can also be displayed by non-WCC code by using the **showEvent** method (or the **publishExternalShowEvents** method when using the PageBus Managed Hub). For information about **showEvents**, see showEvents; for information about **publishExternalShowEvents**, see Using the TIBCO PageBus Managed Hub with WCC Components.

For more information, see pageBusPayloadPublishApp.

Both of these sample applications display a Login dialog. Therefore, you must have a valid user name and password to use these applications.

Also note that an `applicationName.create.war.cmd` file is provided for both of the pageBus sample applications. These file are used to create a WAR file that can be used to deploy the sample application to a runtime node. For more information, see the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Configuration and Customization* guide.

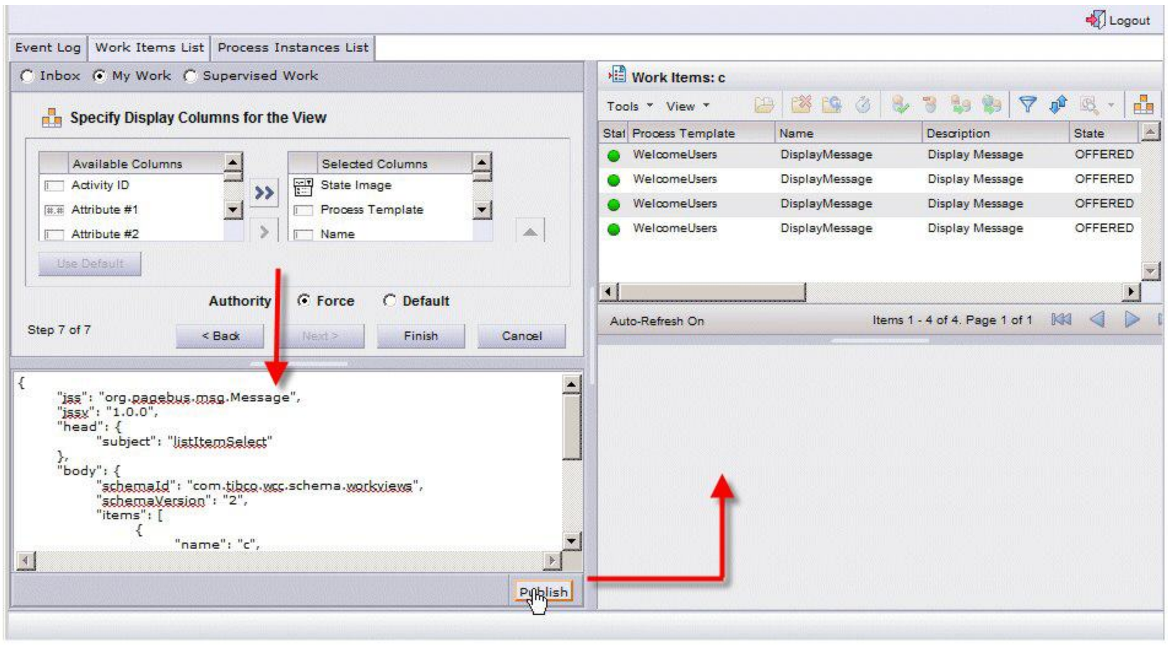
pageBusPayloadHelperApp

This application provides a wizard that guides you through building the desired work item or process instance list, then generates the PageBus event payload.

Note that the pageBusPayloadHelperApp is *not* intended to be opened and the code examined. Use this application *only* to generate events for a work item or process instance list, then use the pageBusPayloadPublishApp to publish the event and render the list (the pageBusPayloadPublishApp *is* intended to be opened and examined as an example of how to publish the event).

Also note that none of the data entry into the wizard in the pageBusPayloadHelperApp application is validated — you are responsible to ensure entered data (e.g., filter strings) are valid. If they are not, the results can be unpredictable.

When the pageBusPayloadHelperApp application is run, a dialog similar to the following is displayed:



```
{
    "iss": "org.pagebus.msg.Message",
    "issv": "1.0.0",
    "head": {
        "subject": "listItemSelect"
    },
    "body": {
        "schemaId": "com.tibco.wcc.schema.workviews",
        "schemaVersion": "2",
        "items": [
            {
                "name": "c",
```

Stat	Process Template	Name	Description	State
●	WelcomeUsers	DisplayMessage	Display Message	OFFERED
●	WelcomeUsers	DisplayMessage	Display Message	OFFERED
●	WelcomeUsers	DisplayMessage	Display Message	OFFERED
●	WelcomeUsers	DisplayMessage	Display Message	OFFERED

The upper-left pane contains the wizard that is used to specify the work item or process instance list, depending on whether the **Work Item List** or **Process Instance List** tab is selected. There is also an **Event Log** tab (which contains the PageBus Event Monitor Component — see Viewing Triggered Events using the PageBus Event Monitor Component for more information) that allows you to see events that are fired in the sample application.

When you click the **Finish** button in the wizard, the PageBus event message payload for the list is displayed in the lower-left pane. Clicking the **Publish** button in the lower-left pane causes the list to be rendered in the right pane. This allows you to see the list to ensure it is what you expected.

You can only generate and publish a single event on either the **Work Item List** or the **Process Instance List** tab. In other words, if you generate an event payload for a work item list, you must restart the application before generating another work item list payload (after clicking the **Publish** button, it, as well as the **Finish** button in the wizard, become disabled, preventing you from generating another event of the same type without restarting the application). You can, however, generate an event payload on one tab, then switch to the other tab and generate an event payload on that tab.

For information about the data in the event payload that is generated by the pageBusPayloadHelperApp, see Event Payloads for the Work Item and Process Instance Lists.

Setting Up the pageBusPayloadHelperApp Application

Procedure

1. Copy the following directory...

   StudioHome\wcc\version\Samples\wccPageBusPayloadApplications\JSXAPPS\pageBusPayloadHelperApp\

   ... to the following directory:

   StudioHome\wcc\version\JSXAPPS\

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.
2. Open the application's configuration file, which is located as follows:

   StudioHome\wcc\version\JSXAPPS\pageBusPayloadHelperApp\config.xml
3. Locate the **ActionProcessors** record in the `config.xml` file.
4. Set the **baseUrl** attribute to the URL of the Action Processor. The string in the **baseUrl** attribute must be in the form:

   ```
   http://Host:Port/bpm/actionprocessor/actionprocessor.servlet
   ```

   where:

   - *Host* is the name or IP address of the machine hosting the BPM runtime.
   - *Port* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

     Note - The **weighting** attribute is not used at this time.

     You might also want to also set the **disable** attribute in the **SessionMonitor** record to "true", which will prevent the application from timing out while you are using it.
5. Save and close the `config.xml` file.
6. Copy the pageBusPayloadHelperApp launch fragment from here...

   StudioHome\wcc\version\Samples\wccPageBusPayloadApplications\pageBusPayloadHelperApp.html

   ... to here:

   StudioHome\wcc\version

Running the pageBusPayloadHelperApp Application

Procedure

1. Execute the pageBusPayloadHelperApp launch fragment:

   ```
      StudioHome\wcc\version\pageBusPayloadHelperApp.html
   ```

The Login dialog is displayed.

2. Enter a valid user name and password, then click **OK**.

   The PageBus Payload Helper App dialog is displayed, with a wizard displayed in the upper-left pane.

3. In the upper-left pane, click on the appropriate tab, as follows:

   - **Event Log** - This tab contains the **PageBus Event Monitor Component**, which displays all events that fire in the sample application. This can be used as a troubleshooting/testing tool.

     For information about the PageBus Event Monitor Component, see Viewing Triggered Events using the PageBus Event Monitor Component.

   - **Work Items List** - This tab is used to specify a work item list, then generate the event payload for that list.

     For information about generating a work item list event payload, see Creating a Work Item List Event Payload.

   - **Process Instances List** - This tab is used to specify a process instance list, then generate the event payload for that list.

     For information about generating a process instance list event payload, see Creating a Process Instance List Event Payload.

Creating a Work Item List Event Payload

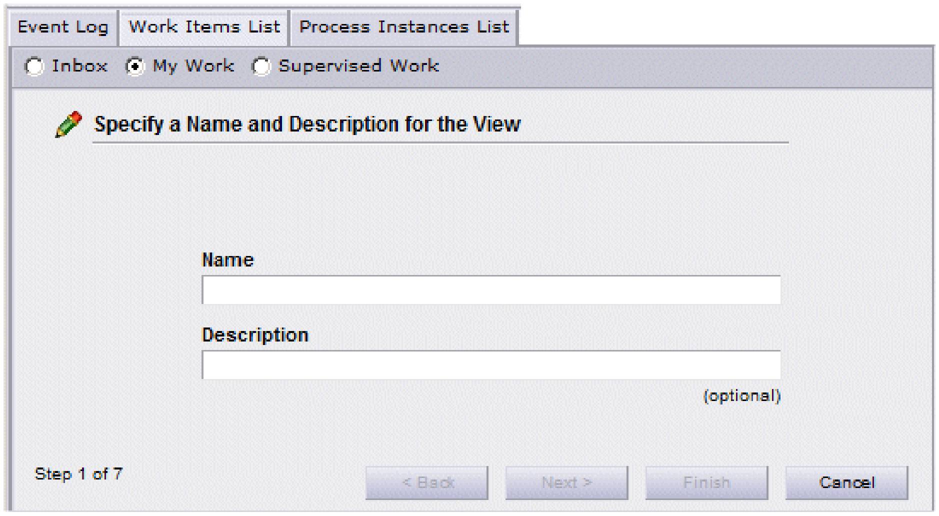
This topic describes using the pageBusPayloadHelperApp application wizard to generate and publish a work item list event payload.

For information about the data in the event payload that is generated by the pageBusPayloadHelperApp, see Event Payloads for the Work Item and Process Instance Lists.

Procedure

1. From the wizard in the upper-left pane, click on the **Work Items List** tab.

   The following dialog is displayed:



2. Choose the appropriate selection at the top of the wizard:

   - **Inbox** - This causes the wizard to generate an “Inbox” work item list, which is technically a “personal” work view (see **My Work** below).

- **My Work** - This causes the wizard to generate a "personal" work view that by default contains work items that have been sent to the logged-in user.
- **Supervised Work** - This causes the wizard to generate a "supervised" work view, which contains work items sent to either an individual resource or an organizational entity that the logged-in user supervises.

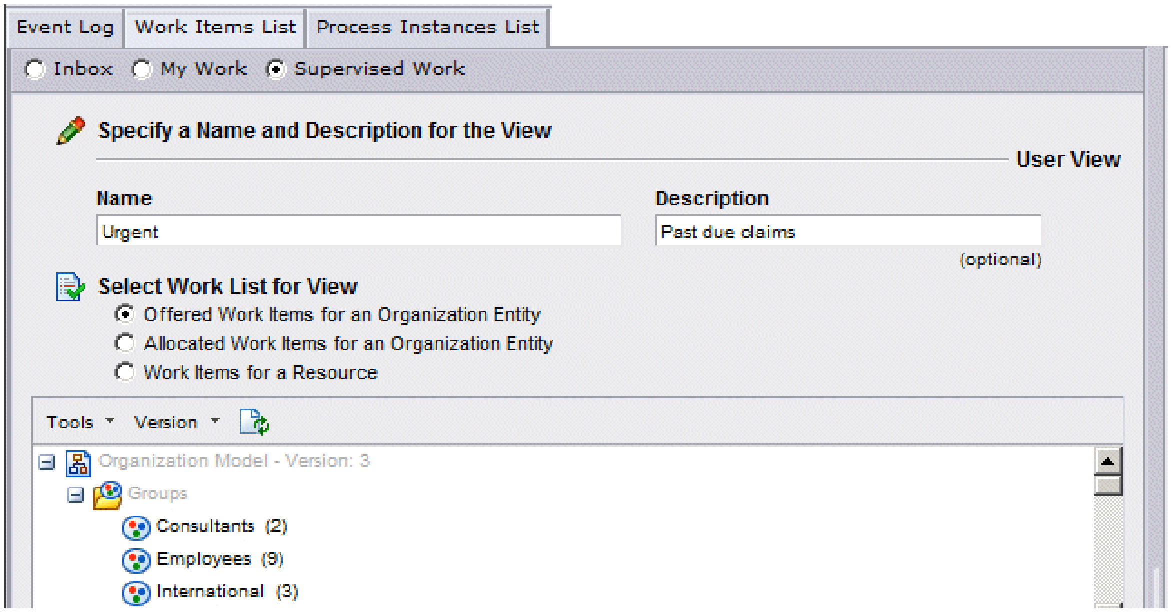
3. Enter a name in the **Name** field, and optionally a description in the **Description** field, for the work view you are defining(1).

   If you are defining a personal work view (Inbox or My Work), you can now click **Next** to advance to the next dialog in the wizard — proceed to step 5.

   If you are defining a supervised work view (Supervised Work), continue on to the next step.

   Once the **Finish** button is enabled (i.e., as soon as a name is entered for personal work views, or a name is entered and a resource or organizational entity is selected for a supervised work view), you can click **Finish** to complete the work view definition and generate the event payload. Default values will be used for all non-specified criteria.

4. Specify the organizational entity or resource for which you are creating the supervised work view, using the following substeps:


Event Log | Work Items List | Process Instances List

Inbox My Work [✓ Supervised Work]

Specify a Name and Description for the View

User View

Name: Urgent

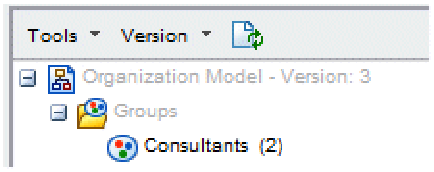
Description: Past due claims (optional)

Select Work List for View
- [✓ Offered Work Items for an Organization Entity]
- Allocated Work Items for an Organization Entity
- Work Items for a Resource

Tools ▾ Version ▾

- Organization Model - Version: 3
  - Groups
    - Consultants (2)
    - Employees (9)
    - International (3)

   a) Ensure that the appropriate Organization Model version is selected. This is shown in the first row of the Organization Model (Version 3 in the following example):


Tools ▾ Version ▾

- Organization Model - Version: 3
  - Groups
    - Consultants (2)

   b) Use the **Version** drop-down list to change the version, if required.

   c) In the **Select Work List for View** section, choose the type of work items that are to appear in the supervised work view, as follows:

[9] You cannot use any of the four following characters in the view name or description: < (less than), > (greater than), & (ampersand), or " (double quote). The application will not allow you to enter any of these characters.

- **Offered Work Items for an Organizational Entity** - This causes the supervised work view to contain only work items that were sent directly to the selected organizational entity, and that still have a state of Offered.

  This type of supervised work view will contain work items that are *offered directly to that entity* — it does *not* contain work items that are offered or allocated to all of the *members* of the chosen organization entity.

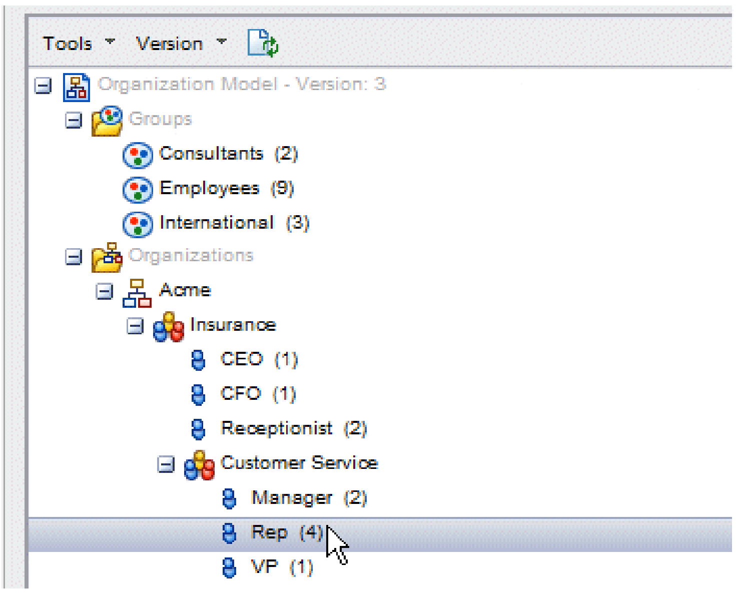
- **Allocated Work Items for an Organizational Entity** - This causes the supervised work view to contain only work items that were originally sent directly to the selected organizational entity,1 but have since been allocated to a specific resource (their state = Allocated). Note that when viewing this type of supervised work view, it does not show you to whom the work item has been allocated. However, you can easily determine that by selecting a work item, then selecting **Open Event Viewer** > **This Work Item**. The allocation event shows to whom the work item was allocated (in the **Resource name** attribute). The **Description** column also shows that information if it has not been changed from the default.

- **Work Items for a Resource** - This causes the supervised work view to contain work items that are currently in the Inbox of a specified resource, offered and allocated.

  If the message "Insufficient rights to view the resource list for a group or position" is displayed when you select **Work Items for a Resource**, it means you don't have the appropriate system action. To view the resource list, you must have the **Resource Admin** (DE.resourceAdmin) system action, which is available at the organization model level. For more information, see the "Configuring User Access" topic in the *TIBCO Workspace Configuration and Customization* guide).

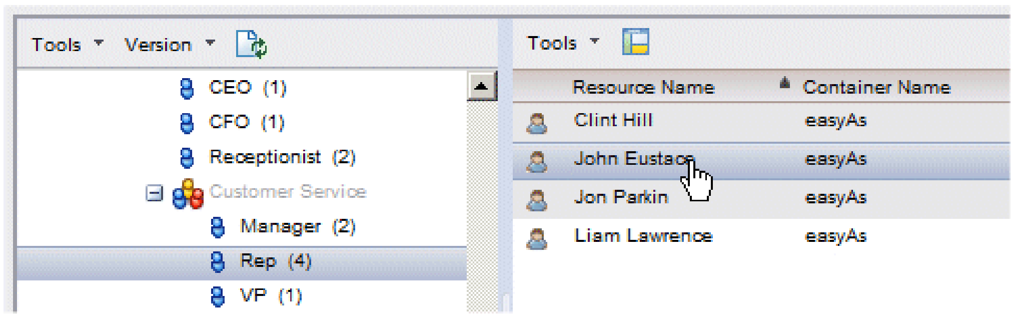
d) Select the desired organizational entity or an individual resource:

- If the supervised work view is to contain work items (either offered or allocated) sent to an organizational entity, select the desired organization unit, position, or group. For example:



  Note that only organizational entities to which you have been given access permission are selectable — all others are grayed out. This permission is granted via the **View Work List** (BRM.viewWorkList) *scoped* system action. For more information, see the "Configuring User Access" topic in the *TIBCO Workspace Configuration and Customization* guide.

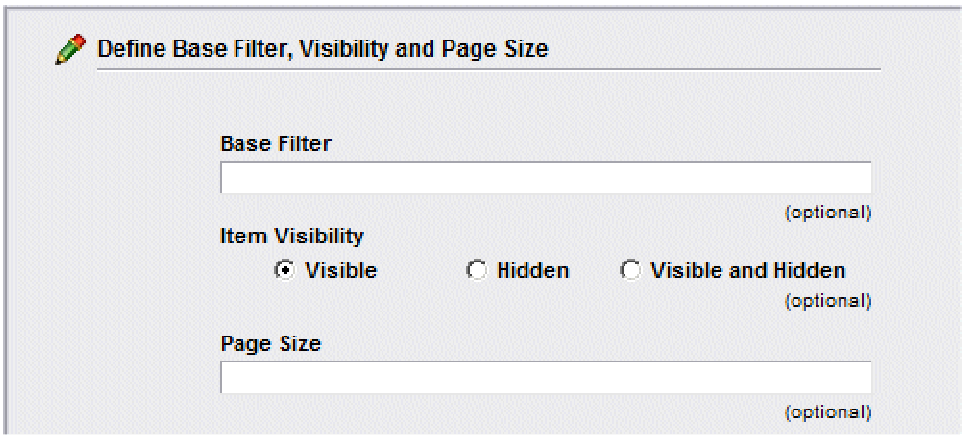
- If the supervised work view is to contain work items sent to an individual resource, you must first select one of the positions to which that resource has been mapped (you cannot select resources from groups, only from positions), then select the resource in the right pane. For example, if you want to create a supervised work view for a resource that has been mapped to the CustomerServiceRepresentative position, select that position, then select the desired resource. For example:



Resource Name	Container Name
Clint Hill	easyAs
John Eustace	easyAs
Jon Parkin	easyAs
Liam Lawrence	easyAs

e) Click **Next** to advance to the next wizard dialog.

5. Specify a base filter, visibility, and page size on the following dialog, then click **Next**:


Define Base Filter, Visibility and Page Size

Base Filter

(optional)

Item Visibility

[✓ Visible] Hidden Visible and Hidden

(optional)

Page Size

(optional)

- **Base Filter** - This is a filter for the work view that is imposed every time the work item list is displayed, but it is *not* visible to the user. If a *refined* filter is specified, either in this wizard or from the Filter dialog after the work item list is displayed, the refined filter is added to the base filter.

  If you specify a base filter, you must enter valid filter syntax in the **Base Filter** field — it is *not* validated. For information about filter syntax, see the "Filtering Lists" topic in the *TIBCO Workspace User's Guide.*

  Also note that when filtering on text, it is case sensitive.

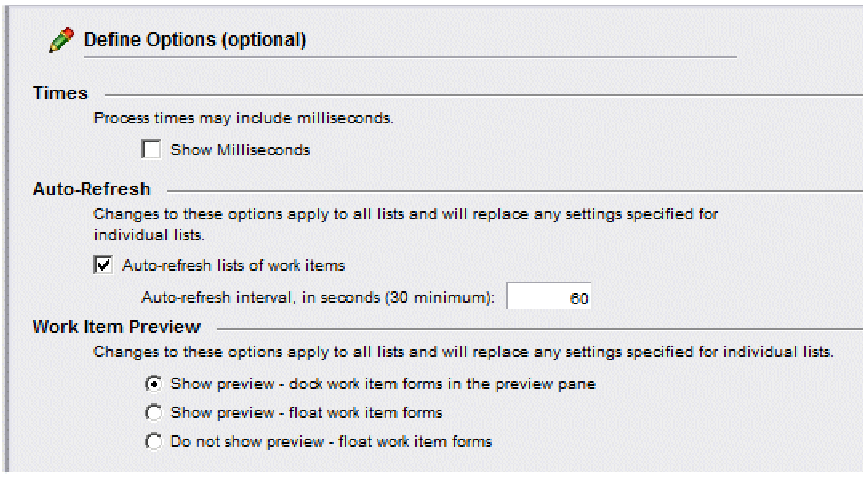
  One thing you can do is advance three dialogs in the wizard, and enter the desired base filter on the Filter dialog (which is where a refined filter is defined). After entering the filter, select **Show Expression** from the **Tools** menu, which displays the created filter syntax on the bottom of the Filter dialog. Unfortunately, the expression cannot be copied, but it does allow you to view the valid syntax.

- **Item Visibility** - These three selections allow you to specify that only visible, only hidden, or both visible and hidden work items be displayed on the created work item list.

  Work items may be hidden for two reasons: 1) They are work items that are associated with a process instance that has been suspended; 2) They are work items that have been pended, that is, a timer has been set to make them hidden until a specified date/time, or for a specified period of time.

- **Page Size** - Specify the number of work items you want on each *page*; the user can step through multiple pages to view all available work items. It defaults to 20.

6. Using the following dialog, specify the Options that are relevant to work item lists, then click **Next**:



Define Options (optional)

Times

Process times may include milliseconds.

[ ] Show Milliseconds

Auto-Refresh

Changes to these options apply to all lists and will replace any settings specified for individual lists.

[✓] Auto-refresh lists of work items

Auto-refresh interval, in seconds (30 minimum): 60

Work Item Preview

Changes to these options apply to all lists and will replace any settings specified for individual lists.

(●) Show preview - dock work item forms in the preview pane

( ) Show preview - float work item forms

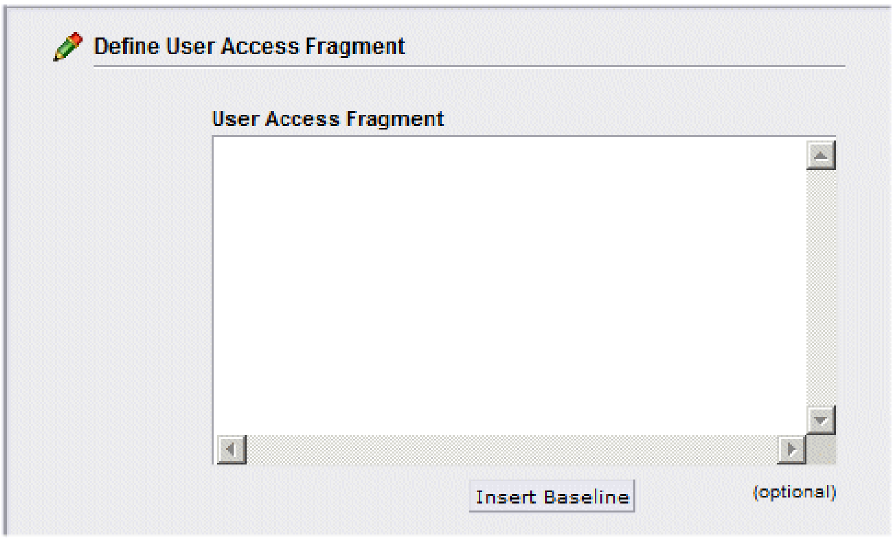
( ) Do not show preview - float work item forms

- **Times** - Check the **Show Milliseconds** box to cause work item times to include milliseconds.
- **Auto-Refresh** - This option specifies whether or not the auto-refresh feature on the created work item list is enabled or disabled by default. When enabled, the work item list is automatically refreshed at a specified interval.

  Check the **Auto-refresh lists of work items** box to enable default auto-refresh. If enabled, specify the auto-refresh interval, in seconds, in the **Auto-refresh interval** field.
- **Work Item Preview** - This option specifies the default setting of the Preview feature on the created work item list, which specifies whether or not work item forms are displayed in the preview pane or in a floating window by default.

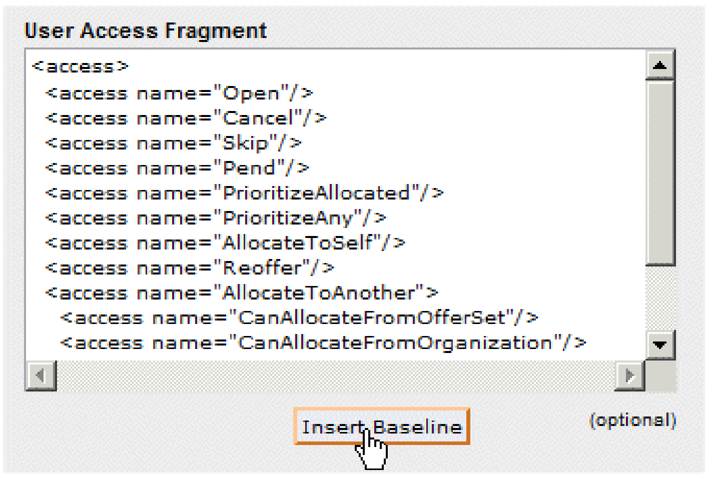
  Choose the appropriate selection as follows:
- **Show preview - dock work item forms in the preview pane** - When the user *selects* (single-clicks) a work item in the list, the work item summary is displayed in the preview pane. When the user *opens* (double-clicks) a work item, the work item form is displayed in the preview pane, replacing the summary information.
- **Show preview - float work item forms** - When the user *selects* (single-clicks) a work item in the list, the work item summary is displayed in the preview pane. When the user *opens* (double-clicks) a work item, the work item form is displayed in a floating window.
- **Do not show preview - float work item forms** - Selecting (single-clicking) a work item has no effect. When the user *opens* (double-clicks) a work item, the work item form is displayed in a floating window.

7. Using the following dialog, specify user access permissions, then click **Next**:



This dialog is used to specify which functions will be available on the created work item list.

You can directly enter the appropriate <access/> elements into the **User Access Fragment** field, or click the **Insert Baseline** button to insert the default user access permissions into the field:



```
<access>
  <access name="Open"/>
  <access name="Cancel"/>
  <access name="Skip"/>
  <access name="Pend"/>
  <access name="PrioritizeAllocated"/>
  <access name="PrioritizeAny"/>
  <access name="AllocateToSelf"/>
  <access name="Reoffer"/>
  <access name="AllocateToAnother">
    <access name="CanAllocateFromOfferSet"/>
    <access name="CanAllocateFromOrganization"/>
```

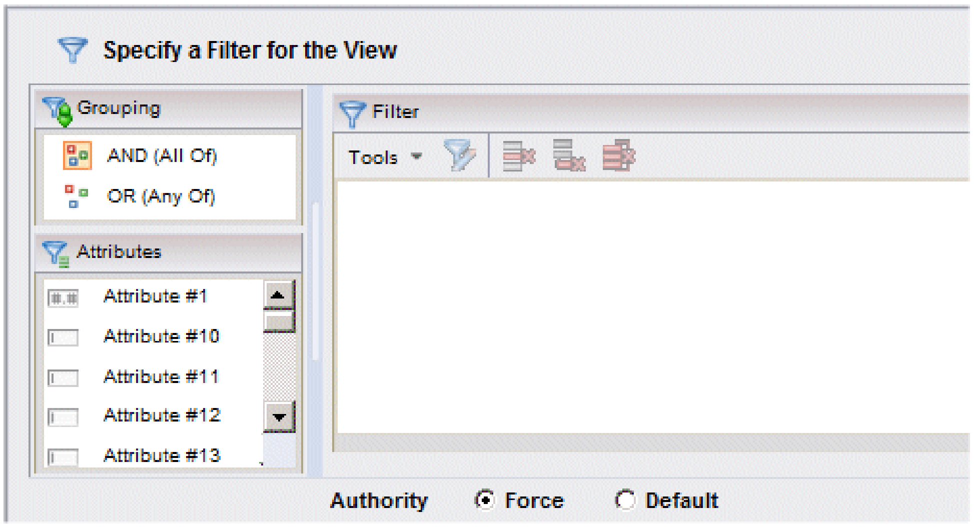
Note that the <access/> elements that are inserted by the **Insert Baseline** button differ somewhat depending on whether you chose **Inbox**, **My Work**, or **Supervised Work**.

Once the baseline is inserted, you can remove access to a particular function on the created work item list by deleting the appropriate line in the fragment. (Again, this is *not* validated — if an invalid XML entry is made, the application may not work properly.)

If you do not enter anything in the **User Access Fragment** field, the default access will still apply on the created work item list, it is just not sent in the event payload.

For more information about user access permissions, see the "Configuring User Access" topic in the *TIBCO Workspace Configuration and Customization* guide.

8. Using the following dialog, specify a filter that is to be applied on the created work item list, then click **Next**:



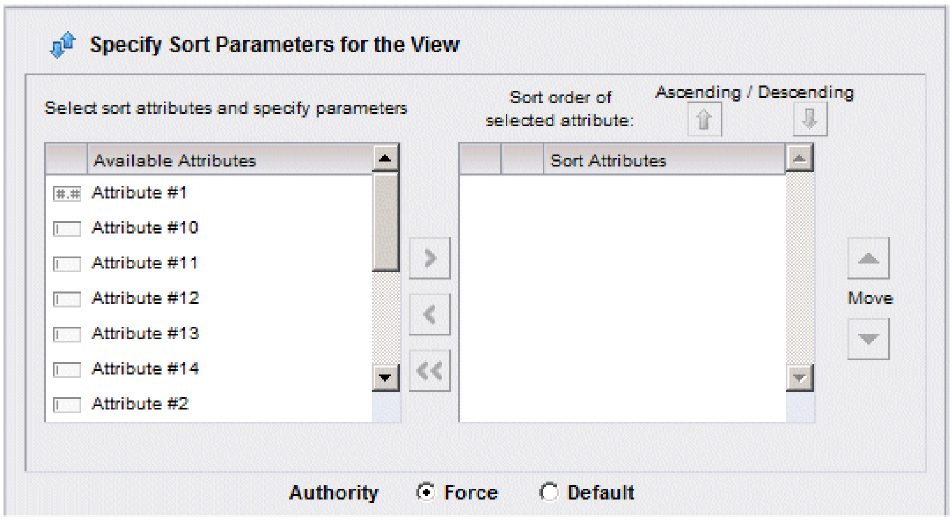
Authority [✓ Force] Default

For complete information about filtering work item lists, see the "Filtering Lists" topic in the *TIBCO Workspace User's Guide.*

The **Authority** selections allow you to specify whether or not to forcibly impose the filter on the list, as follows:

- **Force** - The filter is *always* imposed on the list, regardless if the user had previously set a different filter on the list. In other words, it does not look in user data for a persisted filter on the list.
- **Default** - Prior to the list being displayed, user data is checked to see if the user had previously specified a filter for the list; if there is a persisted filter, it uses that, otherwise it uses the filter specified here (if no filter is specified here, no filter is imposed).

9. Using the following dialog. specify a sort that is to be applied on the created work item list, then click **Next**:



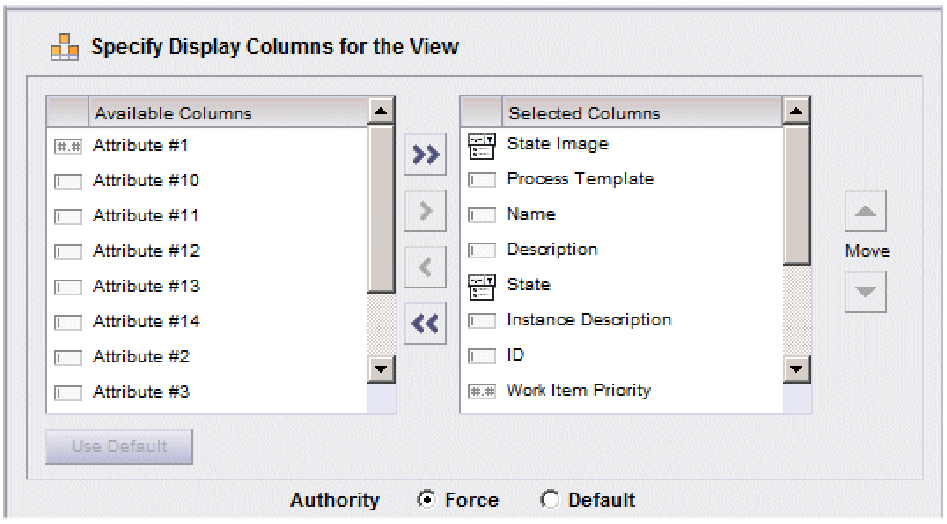
Authority [✓ Force] Default

For complete information about sorting work item lists, see the "Sorting Lists" topic in the *TIBCO Workspace User's Guide.*

The **Authority** selections allow you to specify whether or not to forcibly impose the sort on the list, as follows:

- **Force** - The sort is *always* imposed on the list, regardless if the user had previously set a different sort on the list. In other words, it does not look in user data for a persisted sort on the list.
- **Default** - Prior to the list being displayed, user data is checked to see if the user had previously specified a sort for the list; if there is a persisted sort, it uses that, otherwise it uses the sort specified here (if no sort is specified here, no sort is imposed).

10. Using the following dialog, specify the columns that are to be displayed on the created work item list:



Authority [✓ Force] Default

For complete information about setting columns on work item lists, see the "Working With Work Items" topic in the *TIBCO Workspace User's Guide.*

The **Authority** selections allow you to specify whether or not to forcibly impose columns on the list, as follows:

- **Force** - The specified columns are *always* imposed on the list, regardless if the user had previously set different columns on the list. In other words, it does not look in user data for persisted columns on the list.
- **Default** - Prior to the list being displayed, user data is checked to see if the user had previously specified columns for the list; if there are persisted columns, it uses them, otherwise it uses the columns specified here.

11. Click **Finish** to complete the definition of the work item list.

    The PageBus event message (in the form of a JSON string) that is the result of the list definition is displayed in the lower-left pane.

12. You can publish the event that is created by either:

- clicking the **Publish** button in the pageBusPayloadHelperApp, which displays the defined list in the right pane, or
- pasting the event into the pageBusPayloadPublishApp so that it can be published from that sample application — see pageBusPayloadPublishApp.

You can only generate and publish a single event on either the **Work Item List** or the **Process Instance List** tab. In other words, if you generate an event payload for a work item list, you must restart the application before generating another work item list payload (after clicking the **Publish** button, it, as well as the **Finish** button in the wizard, become disabled, preventing you from generating another event of the same type without restarting the application). You can, however, generate an event payload on one tab, then switch to the other tab and generate an event payload on that tab.

Creating a Process Instance List Event Payload

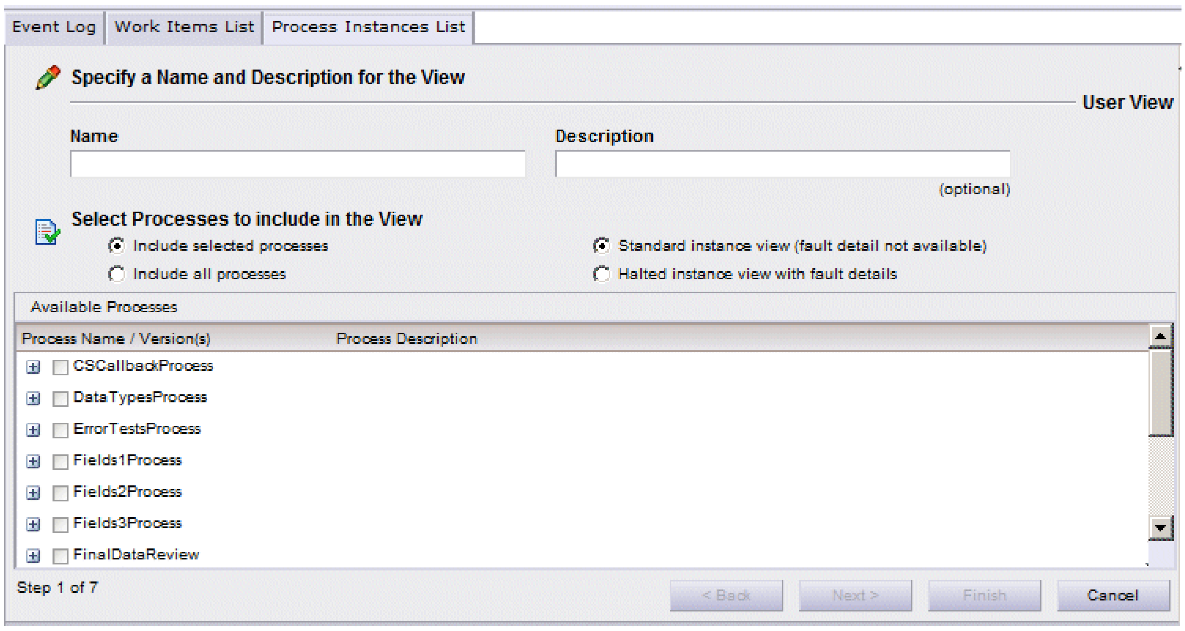
This topic describes using the pageBusPayloadHelperApp application wizard to generate and publish a process instance list event payload.

For information about the data in the event payload that is generated by the pageBusPayloadHelperApp, see Event Payloads for the Work Item and Process Instance Lists.

Procedure

1. From the wizard in the upper-left pane, click on the **Process Instance List** tab.

   The following dialog is displayed:



   This dialog is used to name the new process view, as well as select one or more process templates that are to be included in the view. Note that process template is the term used when referring to the definition of a process. Instances of processes are started by business services. You will be able to see those instances in the view you are creating.

2. Enter a name in the **Name** field, and optionally a description in the **Description** field, for the process instance list you are defining(1).

3. Choose the appropriate selection near the top of the dialog to specify whether you want to choose from the list of process templates that currently exist on the server, or all process templates, current and future. These options are described below:

[10] You cannot use any of the four following characters in the view name or description: < (less than), > (greater than), & (ampersand), or “ (double quote). The application will not allow you to enter any of these characters.

- **Include selected process templates** - Selecting this causes the list of process templates that currently exist on the server to be displayed. This allows you to page through the list and select the desired process templates whose instances you want to appear in the process view you are creating.
- **Include all process templates** - Selecting this causes *all* process templates to be included in the view, which in turn causes *all* instances of those process templates to appear in the process instance list when the view is selected.

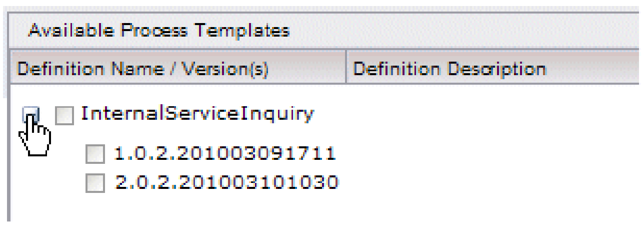
  If you choose this selection, the process view you are creating will include all instances of all process templates that exist when the view is *selected*, not when it was *created*. Note that this is an important distinction from choosing the **Include selected process templates** selection and adding all of the process templates to the view — when that process view is selected, the process instance list will include the instances from all of the process templates that existed when the view was *created*, not when it is *selected*. If new process templates are added to the system, a process view created with the **Include all process templates** selection will include those process templates as well.

  Note that the default "All Instances" process view has **Include all process templates** selected by default.

  If you select **Include all process templates**, the list of process templates on the Select Process Template in the View dialog goes away, since you don't need the list to select from; click **Next** then proceed to step 5.

4. Select the process template(s) whose instances you would like to appear in the process view you are creating, then click **Next**.

   You can expand each of the templates to list all of the versions that are available for that template by clicking on the ⊞ button:



Available Process Templates

Definition Name / Version(s)	Definition Description
☐ InternalServiceInquiry	
☐ 1.0.2.201003091711	
☐ 2.0.2.201003101030	

   You can select process templates you want to include in the process view in the following ways:

   - Clicking the boxes to the left of the version numbers selects only those specific versions.
   - Clicking a top-level box (i.e., next to the template name) causes all versions of the template to automatically be selected. Note that if you select the top-level box, it causes all *future* versions of the template to also be included in the process view.

5. Choose whether you are creating a standard instance view or a halted instance view by selecting one of the following (note that these selections appear on the dialog only if you have the user access authority to define halted instance views), then click **Next**:

   - A *standard instance view* is one in which you will see the normal information about process instances. This type of process view does *not*, however, include details about *halted* processes instances.
   - A *halted instance view* contains attributes that provide information about why a process instance has become halted. This is a special-use process view that is typically only used by supervisory personnel for troubleshooting halted process instances.

     For more information about halted process instances, see the *TIBCO Workspace User's Guide*.

6. On the Define Base Filter and Page Size dialog, specify the following dialog, then click **Next**:

   - **Base Filter** - This is a filter for the process view that is imposed every time the process instance list is displayed, but it is *not* visible to the user. If a *refined* filter is specified, either in this wizard or from the Filter dialog after the process instance list is displayed, the refined filter is added to the base filter.

     If you specify a base filter, you must enter valid filter syntax in the **Base Filter** field — it is *not* validated. For information about filter syntax, see the "Filtering Lists" topic in the *TIBCO Workspace User's Guide.*

     Also note that when filtering on text, it is case sensitive.

     One thing you can do is advance three dialogs in the wizard, and enter the desired base filter on the Filter dialog. After entering the filter, select **Show Expression** from the **Tools** menu, which displays the created filter syntax on the bottom of the Filter dialog. Unfortunately, the expression cannot be copied, but it does allow you to view the valid syntax.

   - **Page Size** - Specify the number of process instances you want on each *page*; the user can step through multiple pages to view all available process instances. It defaults to 20.

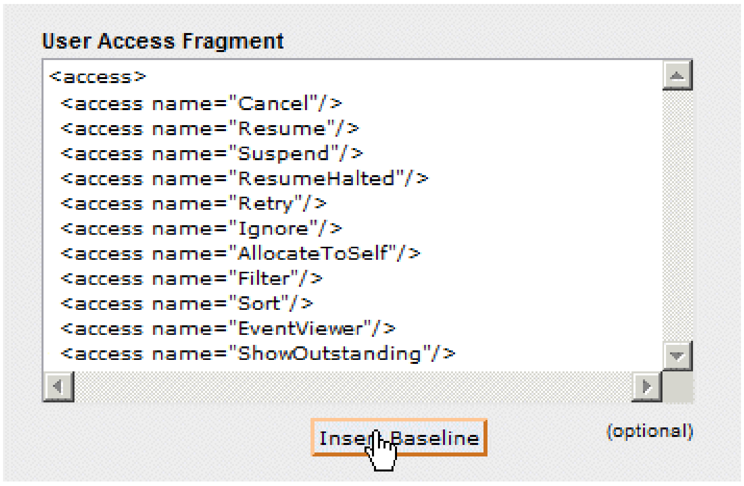
7. On the Define Options dialog, specify the Options that are relevant to process instance lists, then click **Next**:

   - **Times** - Check the **Show Milliseconds** box to cause process instance times to include milliseconds.

8. On the Define User Access Fragment dialog, specify user access permissions, then click **Next**.

   This dialog is used to specify which functions will be available on the created process instance list.

   You can directly enter the appropriate <**access**/> elements into the **User Access Fragment** field, or click the **Insert Baseline** button to insert the default user access permissions into the field:


User Access Fragment

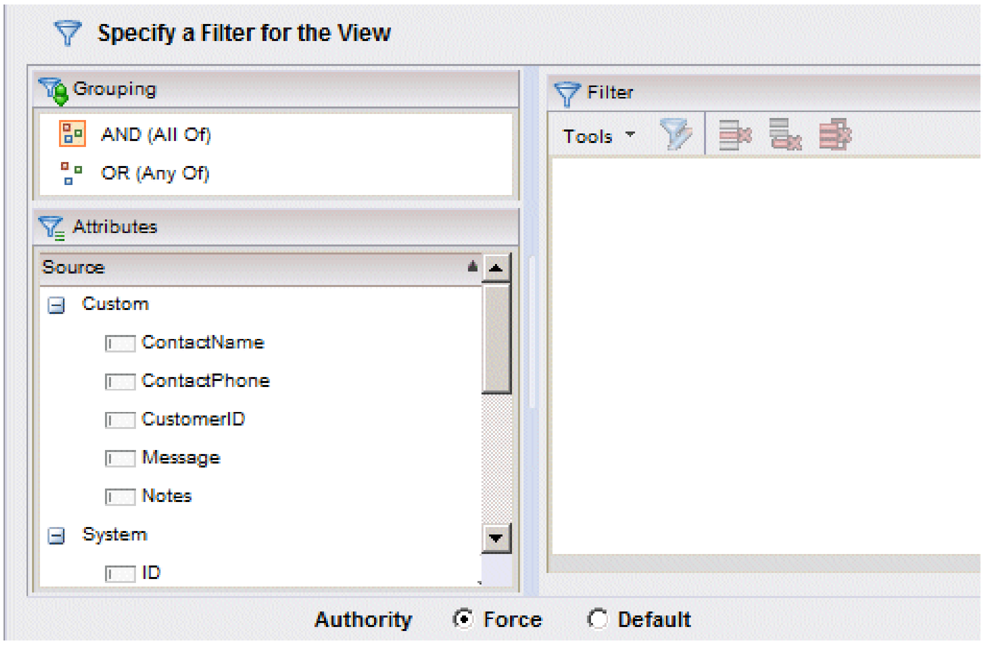
```
<access>
 <access name="Cancel"/>
 <access name="Resume"/>
 <access name="Suspend"/>
 <access name="ResumeHalted"/>
 <access name="Retry"/>
 <access name="Ignore"/>
 <access name="AllocateToSelf"/>
 <access name="Filter"/>
 <access name="Sort"/>
 <access name="EventViewer"/>
 <access name="ShowOutstanding"/>
```

   Once the baseline is inserted, you can remove access to a particular function on the created process instance list by deleting the appropriate line in the fragment. (Again, this is *not* validated — if an invalid XML entry is made, the application may not work properly.)

   If you do not enter anything in the **User Access Fragment** field, the default access will still apply on the created process instance list, it is just not sent in the event payload.

   For more information about user access permissions, see "Configuring User Access" in the *TIBCO Workspace Configuration and Customization* guide.

9. Using the following dialog, specify a filter that is to be applied on the created process instance list, then click **Next**:

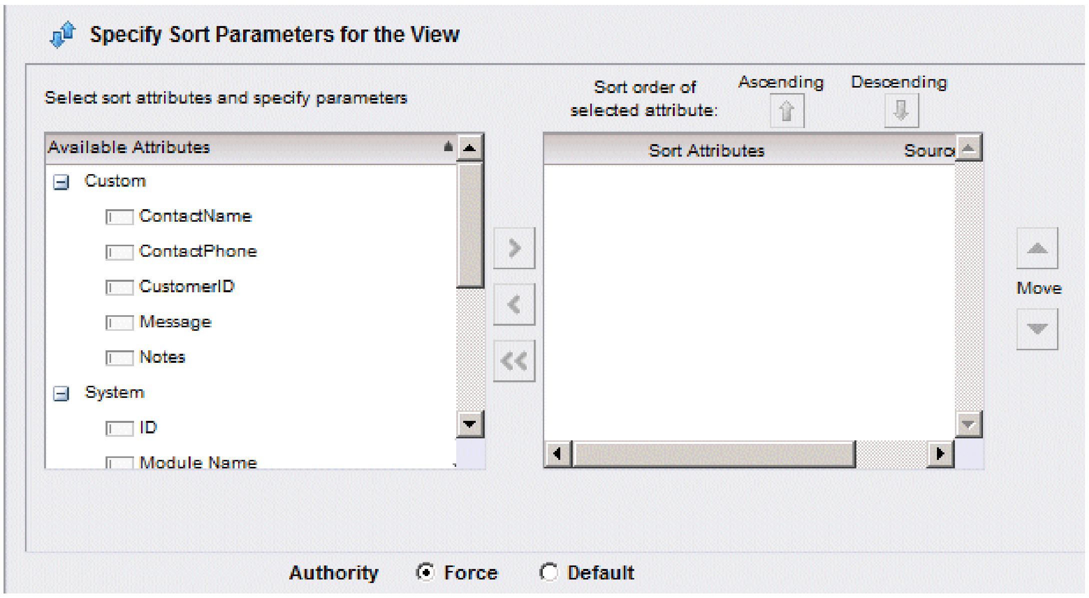


For complete information about filtering process instance lists, see the "Filtering Lists" topic in the *TIBCO Workspace User's Guide.*

The **Authority** selections allow you to specify whether or not to forcibly impose the filter on the list, as follows:

- **Force** - The filter is *always* imposed on the list, regardless if the user had previously set a different filter on the list. In other words, it does not look in user data for a persisted filter on the list.
- **Default** - Prior to the list being displayed, user data is checked to see if the user had previously specified a filter for the list; if there is a persisted filter, it uses that, otherwise it uses the filter specified here (if no filter is specified here, no filter is imposed).

10. Using the following dialog, specify a sort that is to be applied on the created process instance list, then click **Next**:



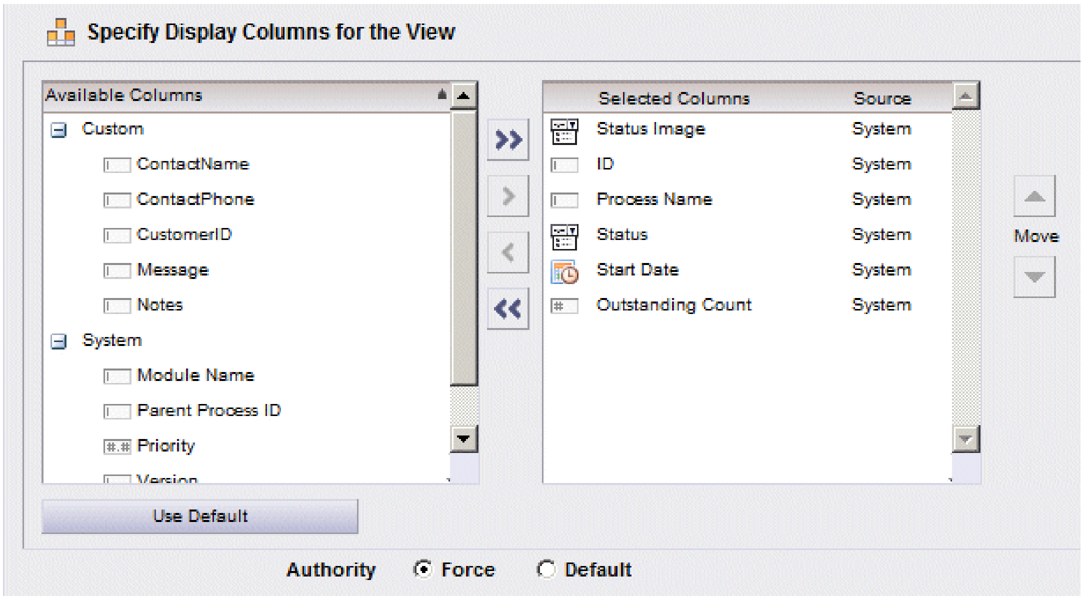
Authority [✓ Force] Default

For complete information about sorting process instance lists, see the *Sorting Lists* chapter in the *TIBCO Workspace User's Guide.*

The **Authority** selections allow you to specify whether or not to forcibly impose the sort on the list, as follows:

- **Force** - The sort is *always* imposed on the list, regardless if the user had previously set a different sort on the list. In other words, it does not look in user data for a persisted sort on the list.
- **Default** - Prior to the list being displayed, user data is checked to see if the user had previously specified a sort for the list; if there is a persisted sort, it uses that, otherwise it uses the sort specified here (if no sort is specified here, no sort is imposed).

11. Using the following dialog, specify the columns that are to be displayed on the created process instance list:



Selected Columns	Source
Status Image	System
ID	System
Process Name	System
Status	System
Start Date	System
Outstanding Count	System

Authority [✓ Force] Default

For complete information about setting columns on process instance lists, see the "Working With Work Items" topic in the *TIBCO Workspace User's Guide.*

The **Authority** selections allow you to specify whether or not to forcibly impose columns on the list, as follows:

- **Force** - The specified columns are *always* imposed on the list, regardless if the user had previously set different columns on the list. In other words, it does not look in user data for persisted columns on the list.
- **Default** - Prior to the list being displayed, user data is checked to see if the user had previously specified columns for the list; if there are persisted columns, it uses them, otherwise it uses the columns specified here.

12. Click **Finish** to complete the definition of the process instance list.

    The PageBus event message (in the form of a JSON string) that is the result of the list definition is displayed in the lower-left pane.

13. You can publish the event that is created by either:

    - clicking the **Publish** button in the pageBusPayloadHelperApp, which displays the defined list in the right pane, or
    - paste the event into the pageBusPayloadPublishApp so that it can be published from that sample application — see pageBusPayloadPublishApp.

    You can only generate and publish a single event on either the **Work Item List** or the **Process Instance List** tab. In other words, if you generate an event payload for a work item list, you must restart the application before generating another work item list payload (after clicking the **Publish** button, it, as well as the **Finish** button in the wizard, become disabled, preventing you from generating another event of the same type without restarting the application). You can, however, generate an event payload on one tab, then switch to the other tab and generate an event payload on that tab.

pageBusPayloadPublishApp

This application is used to publish an event message payload that was generated with the **pageBusPayloadHelperApp** application. It publishes the generated PageBus event, and renders the work item or process instance list that is described in the payload.

For more information, see pageBusPayloadHelperApp.

The intention is for you to open this sample application and examine the code as an example of how to publish an event to the PageBus.

Setting Up the pageBusPayloadPublishApp Application

Procedure

1. Copy the following directory...

   StudioHome`\wcc\`*version*`\Samples\wccPageBusPayloadApplications\JSXAPPS`
   `\pageBusPayloadPublishApp\`

   ... to the following directory:

   StudioHome`\wcc\`*version*`\JSXAPPS\`

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.

2. Open the application's configuration file, which is located as follows:

   StudioHome`\wcc\`*version*`\JSXAPPS\pageBusPayloadPublishApp\config.xml`

3. Locate the **ActionProcessors** record in the `config.xml` file.
4. Set the **baseUrl** attribute to the URL of the Action Processor. The string in the **baseUrl** attribute must be in the form:

   `http://`*`Host`*`:`*`Port`*`/bpm/actionprocessor/actionprocessor.servlet`

   where:

   - *Host* is the name or IP address of the machine hosting the BPM runtime.
   - *Port* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

     Note - The **weighting** attribute is not used at this time.

     You might also want to also set the **disable** attribute in the **SessionMonitor** record to “true”, which will prevent the application from timing out while you are using it.
5. Save and close the `config.xml` file.
6. Copy the pageBusPayloadPublishApp launch fragment from here...

   ```
   StudioHome\wcc\version\Samples\wccPageBusPayloadApplications
   \pageBusPayloadPublishApp.html
   ```

   ... to here:

   ```
   StudioHome\wcc\version\
   ```

Running the pageBusPayloadPublishApp Application

Procedure

1. Execute the pageBusPayloadPublishApp launch fragment:

   ```
   StudioHome\wcc\version\pageBusPayloadPublishApp.html
   ```

   The Login dialog is displayed.
2. Enter a valid user name and password, then click **OK**.

   The PageBus Payload Publish App dialog is displayed.
3. In the left pane, click on the appropriate tab, as follows:

   - **Event Log** - This tab contains the **PageBus Event Monitor Component**, which displays all events that fire in the sample application. This can be used as a troubleshooting/testing tool.

     For information about the PageBus Event Monitor Component, see Viewing Triggered Events using the PageBus Event Monitor Component.
   - **Work Items List** - Click on this tab if you are publishing an event for a work item list.
   - **Process Instances List** - Click on this tab if you are publishing an event for a process instance list.
4. In the left pane, paste the PageBus event payload that you generated with the pageBusPayloadHelperApp application (see pageBusPayloadHelperApp). This must be in the form of a JSON string.
5. Click the **Publish** button.

   The event is published to the PageBus, and the work item list or process instance list that is defined by the event payload is displayed in the right pane.

Event Payloads for the Work Item and Process Instance Lists

The following describes the payload of the events that can be published by non-WCC code using the **Pagebus.publish** method (with the local PageBus) or the **publishPassThruEvent** method (with the PageBus Managed Hub).

For more information, see Non-WCC Code Publishing Events.

These are the payloads that are generated by the **pageBusPayloadHelperApp** application for work item and process instance lists.

Note that if a default is accepted in a field or selection in the **pageBusPayloadHelperApp** application when generating the event, the property for that field or selection may not be included in the event payload.

Work Item List Event Payload

This topic describes the event payload for the work item list.

Data in the PageBus event:

```
topic: "listItemSelect"
message: schemaId = "com.tibco.wcc.schema.workviews"
         schemaVersion = "2"
         items = [
                   {
                     "name": Name of the work item list.
                     "type": Type of work itme list: "INBOX", "MYWORK" or
                             "SUPERVISEDWORK"
                     "description": Description of the work item list.
                     "baseFilter": Filter always imposed on the list; this filter
                              is not visible to the user.
                     "visibility": Whether "visible", "hidden", or "visible and
                              hidden" work items are displayed: "VISIBILEITEMS",
                              "HIDDENITEMS", or "VISIBLEHIDDENITEMS",
                     "pageSize": Number of work items to display per page.
                     "entityType": (Only appears for Supervised work views) The
                              type of entity being supervised: "RESOURCE", "GROUP",
                              "ORGANIZATIONAL_UNIT", or "POSITION".
                     "guid": (Only appears for Supervised work views.) The GUID
                              representing the entity being supervised.
                     "modelVersion": (Only appears for Supervised work views.) The
                              model version used to select the entity being
                              supervised.
                     "sort": Sort order specified for work item list.
                     "column": Columns to display in work item list.
                     "refinementFilter": Filter imposed on the list that the user
                               can see in the Filter dialog.
                     "userAccess": Specifies access to functions on the list.
                     "options": Specifies work item list options to impose on list.
                   }
                 ]
```

Example:

```
topic: "listItemSelect"
message: schemaId = "com.tibco.wcc.schema.workviews"
         schemaVersion = "2"
         items = [
                   {
                     "name": "Urgent",
                     "type": "MYWORK",
                     "description": "Past Due Claims",
                     "baseFilter": "state = ALLOCATED",
                     "visibility": "VISIBLEHIDDENITEMS",
                     "pageSize": "30",
                     "sort": "<sort authority=\"DEFAULT\"><attribute name=
                         \"startDate\" ascending=\"true\"/></sort>",
                     "column": "<columns authority=\"DEFAULT\"><column
                         name=\"StateImage\"/><column name=\"AppName\"/><column
                         name=\"Name\"/><column name=\"Description\"/><column
                         name=\"State\"/><column name=\"AppInstanceDescription\"/>
                         <column name=\"AppInstance\"/><column name=\"Priority\"/>
                         <column name=\"StartDate\"/><column name=
                         \"DeadlineImage\"/><column name=\"EndDate\"/><column
                         name=\"ID\"/><column name=\"Visible\"/></columns>",
                     "refinementFilter": ""<filterCdf authority=\"DEFAULT\"><data
                         jsxid=\"jsxroot\"><record jsxid=\"jsx_1296501534843\"
                         jsximg=\"\" allowDrop=\"insertBefore\" allowDrag=\"yes\"
                         Type=\"COL_NUMERIC\" jsxtype=\"Numeric\" apType=
                         \"COL_NUMERIC\" id=\"4\" description=\"The integer value
                         that denotes the priority of the work item.\" jsxtext=
                         \"Priority  =  50\" Label=\"Priority\" Operator=\"Equal\"
                         Value=\"50\" field=\"priority\" jsxselected=
                         \"1\"/></data></filterCdf>",
                     "userAccess": "<access><access name=\"Open\"/><access name=
                         \"Cancel\"/><access name=\"Skip\"/><access name=
                         \"Pend\"/><access name=\"AllocateToSelf\"/><access name=
                         \"Reoffer\"/><access name=\"AllocateToAnother\"><access
                         name=\"CanAllocateFromOfferSet\"/><access name=
                         \"CanAllocateFromOrganization\"/><access name=
                         \"ShowResourcePreview\"/></access><access name=
                         \"EventViewer\"/><access name=\"Filter\"/><access name=
                         \"Sort\"/><access name=\"AutoRefresh\"/><access name=
                         \"SelectColumns\"/></access>",
                     "options": "<options><option name=\"showMillis\" value=
                         \"true\"/></options>"
                   }
                 ]
```

Process Instance List Event Payload

This topic describes the event payload for the process instance list.

Data in the PageBus event:

```
topic: "listItemSelect"
message: schemaId = "com.tibco.wcc.schema.processViews"
         schemaVersion = "2"
         items = [
                   {
                     "name": Name of the process instance list.
                     "templates": Process templates whose instances are shown in
                              the process instance list.
                     "description": Description of the process instance list.
                     "baseFilter": Filter always imposed on the list; this filter
                              is not visible to the user.
                     "pageSize": Number of process instances to display per page.
                     "sort": Sort order specified for process instance list.
                     "columns": Columns to display in process instance list.
                     "refinementFilter": Filter imposed on the list that the user
                               can see in the Filter dialog.
                     "userAccess": Specifies access to functions on the list.
                     "options": Specifies process instance list options to impose
                               on the list.
                   }
                 ]
```

Example:

```
topic: "listItemSelect"
message: schemaId = "com.tibco.wcc.schema.processviews"
         schemaVersion = "2"
         items = [
                   {
                     "name": "Sales",
                     "templates": "<templates><template process=
                             \"SalesCallbackProcess\" module=\"*\" version=
                             \"*\"/><template process=\"SalesSurveyProcess\"
                             module=\"*\" version=\"*\"/></templates>",
                     "description": "Sales processes",
                     "baseFilter": "INSTANCE.PRIORITY = 200",
                     "pageSize": "25",
                     "sort": "<sort authority=\"FORCE\"><attribute name=
                             \"INSTANCE.START_DATE\" ascending=\"true\"/></sort>",
                     "columns": "<columns authority=\"FORCE\"><column name=
                             \"StatusImage\"/><column name=\"INSTANCE.ID\"/><column
                             name=\"INSTANCE.NAME\"/><column name=
                             \"INSTANCE.STATUS\"/><column name=
                             \"INSTANCE.START_DATE\"/><column name=
                             \"INSTANCE.PRIORITY\"/><column name=
                             \"INSTANCE.WAITING_WORK_COUNT\"/></columns>",
                     "refinementFilter": "<filterCdf authority=\"FORCE\"><data
                             jsxid=\"jsxroot\"><record jsxid=\"jsx_1296510460021\"
                             jsximg=\"\" allowDrop=\"insertBefore\" allowDrag=
                             \"yes\" Type=\"COL_STATE\" apType=\"string\" jsxtype=
                             \"State\" jsxtext=\"Instance Status  =  Active\"
                             Label=\"Instance Status\" Operator=\"Equal\"
                             Value=\"Active\" field=\"INSTANCE.STATUS\" source=
                             \"System\" jsxselected=\"1\" ExpressionValue=
                             \"'ACTIVE'\"/></data></filterCdf>",
                     "userAccess": "<access><access name=\"Cancel\"/>
                             <access name=\"Resume\"/>
                             <access name=\"Suspend\"/>
                             <access name=\"ResumeHalted\"/>
                             <access name=\"Retry\"/>
                             <access name=\"Ignore\"/>
                             <access name=\"AllocateToSelf\"/>
                             <access name=\"Filter\"/>
                             <access name=\"Sort\"/>
                             <access name=\"EventViewer\"/>
                             <access name=\"ShowOutstanding\"/>
                             <access name=\"ShowOutstandingAdmin\"/>
                             <access name=\"SelectColumns\"/>
                             <access name=\"ShowCustomAttributes\"/>
                             </access>",
                     "options": "<options><option name=\"showMillis\" value=
                         \"true\"/></options>"
                   }
                 ]
```

Subscribing to an Event Published by Non-WCC Code

When non-WCC code publishes an event to display a work item or process instance list, the WCC client application on the other side must subscribe to that event.

To accomplish this, you must do the following:

- Create an *event definition file*, which identifies the PageBus topic (which includes the event) that is published by the non-WCC code.
- Open the WCC application in TIBCO General Interface Builder and *import* the event definition file so that the WCC client can subscribe to the event (importing the event definition file allows the Event Editor to see the event so that a WCC client can subscribe to it).

Create an Event Definition File

The event definition file that identifies the topic that is published to the PageBus must be created manually.

An example file is provided with the simpleProcessInstancePreview application (which is part of the wccPassThruManagedHub sample application). It can be found here:

`StudioHome\wcc\version\Samples\wccPassThruManagedHub\JSXAPPS`
`\simpleProcessInstancePreview\defs\applicationEvents.xml`

This example identifies the topic, which includes the event name, that is published if you are using the wccPassThruManagedHub sample application:

```
<?xml version="1.0"?>
<app model="com.tibco.wcc.wccPassThruManagedHub">
   <prototype model="sampleAppPrototype">
```

```
      <component model="workviews">
         <event name="listItemSelect" schemaId="com.tibco.wcc.schema.workviews"/>
      </component>
      <component model="processviews">
         <event name="listItemSelect" schemaId="com.tibco.wcc.schema.processViews"/>
      </component>
   </prototype>
</app>
```

You can make a copy of this file and modify it to fit your needs. It must be in the form:

```
<?xml version="1.0"?>
<app model="com.tibco.wcc.appModelName">
   <prototype model="prototypeModelName">
      <component model="componentModelName">
         <event name="listItemSelect"
schemaId="com.tibco.wcc.schema.componentModelName"/>
      </component>
   </prototype>
</app>
```

where:

- *appModelName* - This is normally the application model name. It can be any name you want when you are creating your own event definition file.
- *prototypeModelName* - This is normally the model name given to the prototype. This can also be any name you want.
- *componentModelName* - This is normally the model name of the component. When you are creating an event definition file for the purpose of displaying a work item list or a process instance list, use one of the following for component model name:
- **workviews** - Use this if you are displaying a work item list.
- **processviews** - Use this if you are displaying a process instance list.

  The reason you are using one of these names is because these are the components whose schema is expecting a **listItemSelect** event. You can technically use any name you want here, but when you attempt to subscribe to the **listItemSelect** event, a dialog is displayed telling you that you are subscribing to an event that the component is not expecting; you can continue to subscribe even if this dialog is displayed.

Notice the event name is **listItemSelect**, which renders the work item list or process instance list, depending on the payload that was passed when the **publish** method was called.

An example is show below:

```
<?xml version="1.0"?>
<app model="com.tibco.wcc.MyApp">
   <prototype model="MyAppsPrototype">
      <component model="workviews">
         <event name="listItemSelect" schemaId="com.tibco.wcc.schema.workviews"/>
      </component>
   </prototype>
</app>
```

This example would require that the topic the non-WCC code publishes to the PageBus be:

```
com.tibco.wcc.myApp.MyAppsPrototype.workviews.listItemSelect
```

The WCC client will then import the event definition file using the TIBCO General Interface Builder Event Editor, and subscribe to the event — see Importing the Event Definition File and Subscribing to the Event.

After manually creating the event definition file, save it in the directory structure of your WCC application. The name you give the file is also arbitrary; as long as you can point to it when importing it. A suggested file name/path is:

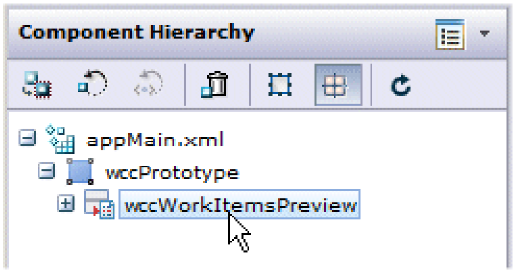
```
...\JSXAPPS\YourAppName\defs\applicationEvents.xml
```

Importing the Event Definition File and Subscribing to the Event

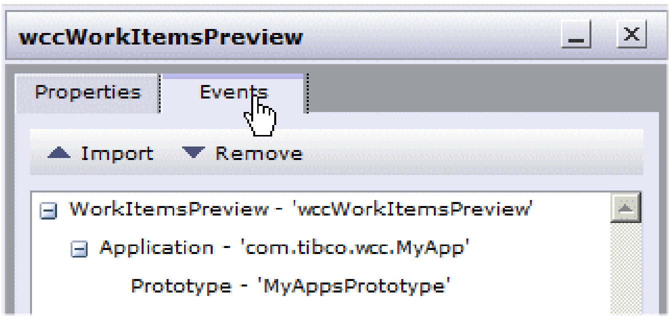
After you have manually created the event definition file, you must import it using TIBCO General Interface Builder, then subscribe to the event.

Procedure

1. Open your client application in TIBCO General Interface Builder.
2. In the Component Hierarchy, select the prototype in your client application:

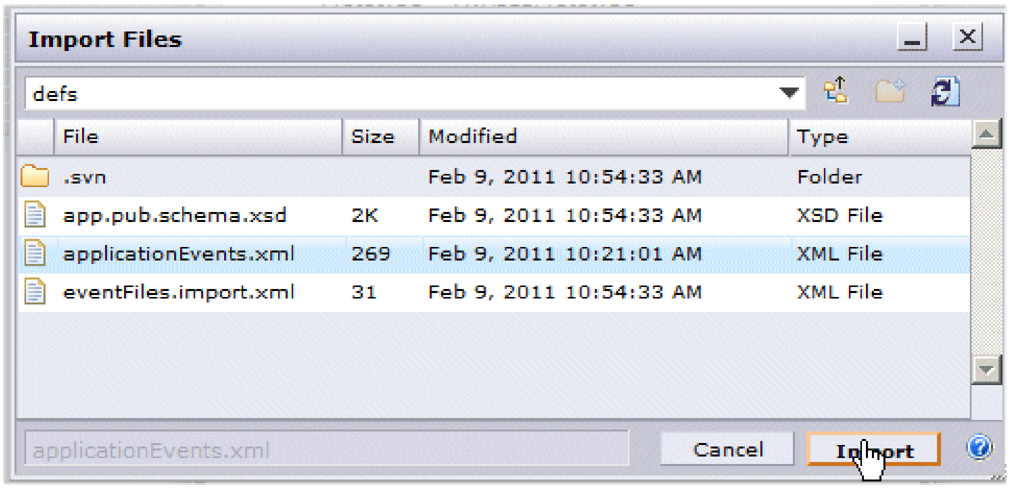


3. Click the button in General Interface Builder's task bar. The Properties/Events editor is displayed.
4. Click on the **Events** tab:



WorkItemsPreview - 'wccWorkItemsPreview'
Application - 'com.tibco.wcc.MyApp'
Prototype - 'MyAppsPrototype'

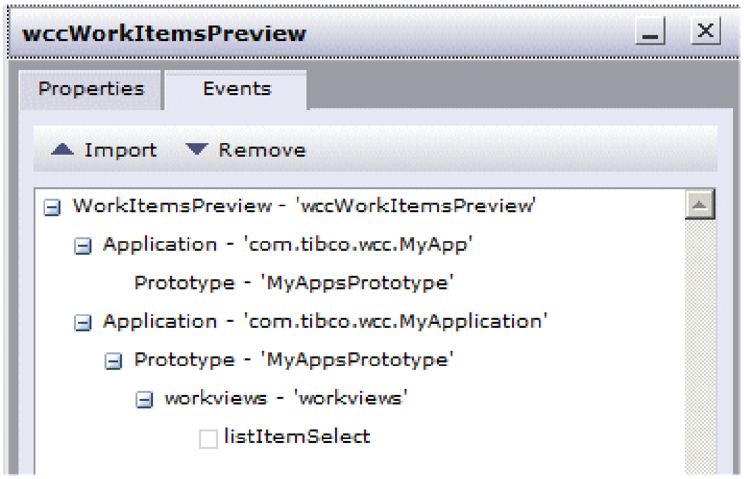
5. Click on the **Import** tab. The Import Files dialog is displayed.
6. Navigate to the directory in which you saved the event definition file, select it, then click **Import**:



File	Size	Modified	Type
.svn		Feb 9, 2011 10:54:33 AM	Folder
app.pub.schema.xsd	2K	Feb 9, 2011 10:54:33 AM	XSD File
applicationEvents.xml	269	Feb 9, 2011 10:21:01 AM	XML File
eventFiles.import.xml	31	Feb 9, 2011 10:54:33 AM	XML File

applicationEvents.xml

The event defined in the event definition file is imported in the Event Editor so you can now subscribe to that event:



WorkItemsPreview - 'wccWorkItemsPreview'
  Application - 'com.tibco.wcc.MyApp'
    Prototype - 'MyAppsPrototype'
  Application - 'com.tibco.wcc.MyApplication'
    Prototype - 'MyAppsPrototype'
      workviews - 'workviews'
        listItemSelect

7. Double-click the **listItemSelect** event, then click **Commit** to complete the subscription.

Using the TIBCO PageBus Managed Hub with WCC Components

If WCC components are being isolated in an iframe, and they are communicating with non-WCC code in another iframe, they must use the TIBCO PageBus Managed Hub to publish or subscribe to events between the components.

A class (**IframeContainer**) is provided with Workspace that contains methods that can be used to simplify the use of the PageBus Managed Hub.

This topic describes the methods available when communicating between components using the PageBus Managed Hub.

Also note that a sample application is provided that illustrates the use of these methods when using the PageBus Managed Hub. For information, see Managed Hub Sample Applications.

For additional information about the TIBCO PageBus Managed Hub, see the *TIBCO PageBus Developer's Guide*.

com.tibco.wcc.components.IframeContainer

The **IframeContainer** class contains the following methods that can be used to contain a WCC application in an iframe:

- **publishExternalLogin** - Publishes an **externalLogin** event to the TIBCO PageBus Managed Hub.
```
publishExternalLogin();
```
- **publishExternalShowEvents** - Publishes a **externalShowEvents** event to the TIBCO PageBus Managed Hub. This causes a temporary event view to be added to the event view list, according to the passed-in parameter values.
```
   publishExternalShowEvents(eventLinkId,
                             substitutionDataXml,
                             menuText,
                             eventLinkXml);
```
  where:
  - *eventLinkId* (String) - The <**links**> element **messageId** attribute value to match. (For information about event links and the <**links**> element, see the *Customizing Events* chapter in the *TIBCO Workspace Configuration and Customization Guide.)*

  - *substitutionDataXml* (String) - The XML containing the attributes that are substituted into the <**link**> element that is used. The root element name is record: <record att1="val1" att2="val2"/>
  - *menuText* (String) (Optional) - If there are multiple <**link**> elements, the **menuText** value is used to locate a specific one. If null, the first link is used.
  - *eventLinkXml* (String) (Optional) - The XML that defines the event links. If this is null, the application-configured **eventLinks** XML is used.

- **publishSetAppLocale** - Publishes a **setAppLocale** event to the TIBCO PageBus Managed Hub to notify application components of the locale change so they can reload and repaint as necessary to reflect the new language.

```
setAppLocale(localeKey);
```

  where:

  - *localeKey* - (String) The lowercase, two-letter ISO-639 language code (ll), and optionally the uppercase (CC), two-letter ISO-3166 country code if the locale is country specific. If the country code is included, it must be separated from the language code with an underscore, in the form "ll_CC" (e.g., "es_MX" for Mexican Spanish).

    For a list of language codes, visit the following web site:

    `http://www.loc.gov/standards/iso639-2/langhome.html`

    For a list of country codes, visit the following web site:

    http://www.iso.org/iso/english_country_names_and_code_elements

- **subscribeAppPostLoad** - Creates a TIBCO PageBus Managed Hub subscription to the **appPostLoad** event.

```
   subscribeAppPostLoad(onData,
                        scope,
                        onComplete,
                        subscriberData);
```

  where:

  - *onData* (Function) Callback function that is invoked whenever an **appPostLoad** event is published.
  - *scope* (Object) Optional. When onData callback or onComplete callback is invoked, the JavaScript "this" keyword refers to this scope object. If no scope is provided, the default is window.
  - *onComplete* (Function) Optional. Callback function that is invoked to tell the client application whether the subscribe operation succeeded or failed.
  - *subscriberData* (Any) Optional. The client application provides this data, which is handed back to the client application in the subscriberData parameter of the onData and onComplete callback functions.

- **subscribeLoginComplete** - Creates a TIBCO PageBus Managed Hub subscription to the **loginComplete** event.

```
   subscribeLoginComplete(onData,
                          scope,
                          onComplete,
                          subscriberData);
```

  where:

  - *onData* (Function) Callback function that is invoked whenever a **loginComplete** event is published.
  - *scope* (Object) Optional. When onData callback or onComplete callback is invoked, the JavaScript "this" keyword refers to this scope object. If no scope is provided, the default is window.

  - *onComplete* (Function) Optional. Callback function that is invoked to tell the client application whether the subscribe operation succeeded or failed.
  - *subscriberData* (Any) Optional. The client application provides this data, which is handed back to the client application in the subscriberData parameter of the onData and onComplete callback functions.

- **publishPassThruEvent** - Publishes a pass-through event (**passThruEvent**) that traverses the managedHub where the subscribing client subscribes to the event and displays either a work item list or process instance list, depending on the payload in the published event.

  ```
  publishPassThruEvent(pageBusSubject,
                       pageBusMessage);
  ```

  where:

  - *pageBusSubject* (String) Topic to publish on the client's local PageBus. (Note - The latest release of the PageBus now refers to "subject" as "topic".)
  - *pageBusMessage* (String) Message to publish on the client's local PageBus.

    For an example of usage, see wccPassThruManagedHub Sample Application.

com.tibco.wcc.base.Application

The **Application** class contains the following methods that can be used when displaying a WCC application in an iframe (although they can be used with non-iframe applications also):

- **isExternalLogin** - Returns the value of the **isExternalLogin** property, which indicates whether or not there is an authenticated login by an external application.

  ```
  isExternalLogin();
  ```

- **initializeExternalLogin** - Initializes the application after login is completed by an external application.

  This method must be called if the login is done by an external application. However, it must *not* be called if the login is done by the current application (i.e., by the **wcc.Login** component or the **app.login()** method, as these call the **initializeExternalLogin** method).

  Note that calling this method has no affect if the user is already logged in.

  ```
  initializeExternalLogin();
  ```

Managed Hub Sample Applications

Two sample applications are also provided that illustrate the use of the methods in the **IframeContainer** class.

They are:

- **Login Managed Hub Application** — see wccLoginManagedHub Sample Application.
- **PassThru Managed Hub Application** — see wccPassThruManagedHub Sample Application.

Note that an `applicationName.create.war.cmd` file is provided for each of the ManagedHubSample applications. These file are used to create a WAR file that can be used to deploy the sample application to a runtime node. For more information, see the "Deploying an Application After Customizing" topic in the *TIBCO Workspace Configuration and Customization* guide.

wccLoginManagedHub Sample Application

The **wccLoginManagedHub** sample application illustrates how to use the methods in the **IframeContainer** class.

With this class, you can:

- Initialize a login for a WCC application running in an iframe, by utilizing a login from a previous login session.
- Display a component (e.g., the Organization Browser) in an iframe.
- Publish an **externalShowEvents** event, which allows an external application to add a temporary event view to the event view list.

The sample is set up to run on a web server at the following URLs:

```
http://wccloginmanagedhubmainsource:9090/wccLoginManagedHub
http://wccloginmanagedhubiframesource:9090/wccLoginManagedHub
```

Note that two URL host names are used in this example, specifically to illustrate the TIBCO PageBus Managed Hub iframe setup:

```
http://wccloginmanagedhubmainsource:9090
http://wccloginmanagedhubiframesource:9090
```

This sample application can be run on a single computer running the web server by setting the host names in the computer's `/etc/hosts` file (`C:\WINDOWS\System32\drivers\etc\hosts` on Windows):

- 127.0.0.1 wccloginmanagedhubmainsource
- 127.0.0.1 wccloginmanagedhubiframesource

Since this is a local alias, the web browser must be running on the same computer as the web server for this setup to work.

Setting Up the wccLoginManagedHub Application

Procedure

1. Add the Workspace framework files to the web server path.

   For example, assuming Tomcat as the web server, copy the following files and directories:

   ```
      - StudioHome\wcc\ver\brand.xml
      - StudioHome\wcc\ver\close.html
      - StudioHome\wcc\ver\externalform.htm
      - StudioHome\wcc\ver\jsx3.gui.window.html
      - StudioHome\wcc\ver\logger.xml
      - StudioHome\wcc\ver\addins\
      - StudioHome\wcc\ver\dojo-toolkit\
      - StudioHome\wcc\ver\JSX\
      - StudioHome\wcc\ver\JSXAPPS\
   ```

   ... to the following directory:

   ```
      - TomcatHome\webapps\wccLoginManagedHub\
   ```

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *ver* is the version number of Workspace that was installed with TIBCO Business Studio.
   - *TomcatHome* is the Tomcat installation directory.

2. Copy the root HTML files needed by the sample application.

   Copy these files:

   ```
      - StudioHome\wcc\ver\samples\wccLoginManagedHub\IframeContainerTest.html
      - StudioHome\wcc\ver\samples\wccLoginManagedHub\simpleEventViewer.html
      - StudioHome\wcc\ver\samples\wccLoginManagedHub\simpleLogin.html
      - StudioHome\wcc\ver\samples\wccLoginManagedHub\simpleOrgBrowser.html
   ```

   ... to the following directory:

   ```
      - TomcatHome\webapps\wccLoginManagedHub\
   ```

3. Copy application directories for the sample application.

   Copy these directories:

   ```
     - StudioHome\wcc\ver\samples\wccLoginManagedHub\JSXAPPS\simpleEventViewer\
     - StudioHome\wcc\ver\samples\wccLoginManagedHub\JSXAPPS\simpleLogin\
     - StudioHome\wcc\ver\samples\wccLoginManagedHub\JSXAPPS\simpleOrgBrowser\
   ```

   ... to the following directory:

   ```
     - TomcatHome\webapps\wccLoginManagedHub\JSXAPPS\
   ```

4. Configure the **baseUrl** for the Action Processor in the sample application's configuration files:
   a) Open the `config.xml` file in the following directories:

   ```
     - TomcatHome\webapps\wccLoginManagedHub\JSXAPPS\simpleEventViewer\
     - TomcatHome\webapps\wccLoginManagedHub\JSXAPPS\simpleLogin\
     - TomcatHome\webapps\wccLoginManagedHub\JSXAPPS\simpleOrgBrowser\
   ```

   b) Locate the **Action Processor** record and set the **baseUrl** attribute to point to the location of your Action Processor. The string in the **baseUrl** attribute must be in the form:

   ```
         http://Host:Port/bpm/actionprocessor/actionprocessor.servlet
   ```

   where:

   - *`Host`* is the name or IP address of the machine hosting the BPM runtime.
   - *`Port`* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

     Note - The **weighting** attribute is not used at this time.

     For example:

Result

```
<record jsxid="ActionProcessors" type="workspace">
    <ActionProcessor
       weighting="100"
       baseUrl="http://Austin:8080/bpm/actionprocessor/actionprocessor.servlet">
    </ActionProcessor>
</record>
```

Executing the wccLoginManagedHub Application

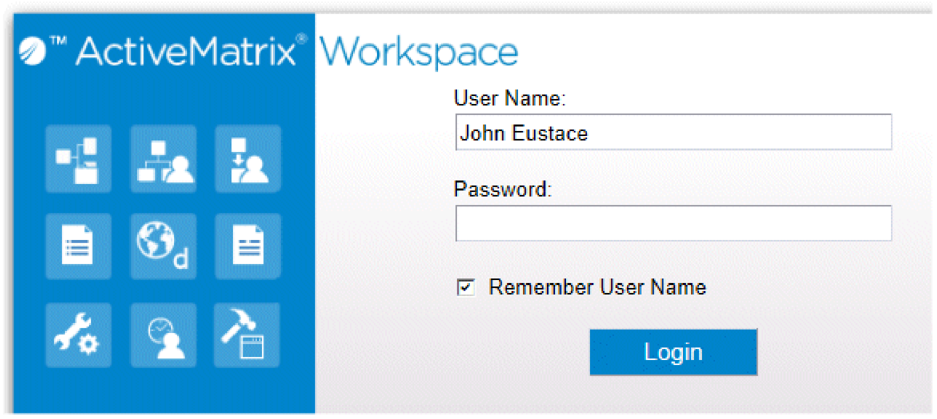
After setting up the **wccLoginManagedHub** application, and starting the web server, you can execute the sample application using the URLs described in this topic.

There are four URLs used in this sample application. They are each described below.

simpleLogin URL

```
http://wccloginmanagedhubmainsource:9090/wccLoginManagedHub/simpleLogin.html
```

This URL opens the simpleLogin WCC application, which displays the **Login** component:



The **postLogin** function for the simpleLogin sample application has been modified to open another browser window that points to the `IframeContainerTest.html` file, as follows:

```
classProto.postLogin = function() {
   this.jsxsuper();
   var extUrl =    'http://wccloginmanagedhubmainsource:9090/    wccLoginManagedHub/
IframeContainerTest.html';
   var windowRef = window.open(extUrl, 'TestWindow');
};
```

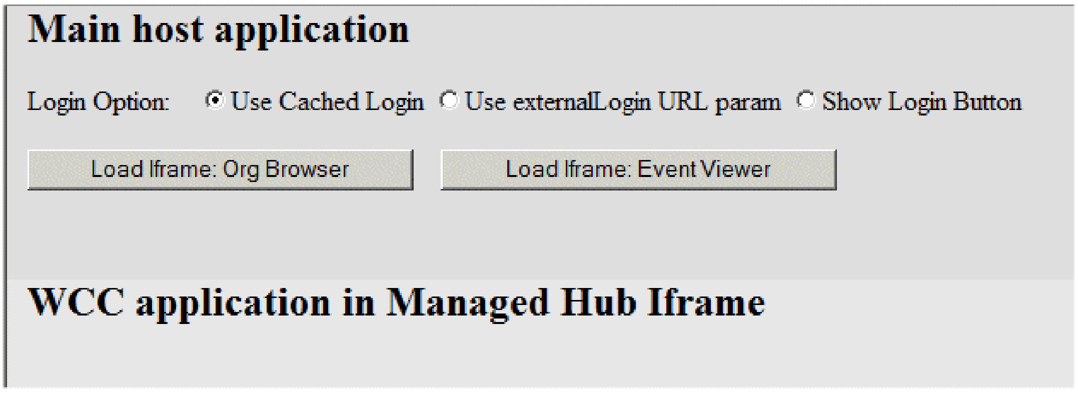
See:

```
...\Samples\wccLoginManagedHub\JSXAPPS\simpleLogin\application\js\Application.js
```

IframeContainerTest URL

http://wccloginmanagedhubmainsource:9090/wccLoginManagedHub/IframeContainerTest.html

This URL, which is launched from the **postLogin()** function in the simpleLogin application, displays a new browser window that allows you to choose a login option, as well as choose whether to load either an Organization Browser component or an EventViews/EventViewer component in an iframe:


Main host application

Login Option: (●) Use Cached Login ( ) Use externalLogin URL param ( ) Show Login Button

WCC application in Managed Hub Iframe

The **Login Option** selections are used to choose the login option, as follows:

- Use Cached Login - This option causes the publishExternalLogin() function to be called, which causes the login to be cached by the ManagedHub.

  WCC applications are loaded into the IframeContainer asynchronously. If the **iFrameContainer.publishExternalLogin** method is called before the WCC application completes loading, the PageBus message is cached and the event is received by the WCC application as soon as it completes loading.

- Use externalLogin URL Param - This option causes “?externalLogin=true” to be appended to the URL. This results in the application being initialized with the user name of the currently authenticated HTTP session.

- Show Login Button - This option causes the Publish External Login to Iframe button to be displayed when either the Load Iframe: Org Browser or Load Iframe: Event Viewer button is clicked.

  The **Publish External Login to Iframe** button calls the **publishExternalLogin()** function to complete the login.

The buttons below the **Login Option** selections are used to create the iFrameContainer, as follows:

- **Load Iframe: Org Browser** - This button creates a `com.tibco.wcc.components.IframeContainer` that is set up to load the **simpleOrgBrowser** WCC application.

  If either the **Use Cached Login** or **Use externalLogin URL Param** login option had been selected, the IframeContainer is directly loaded with the **simpleOrgBrowser** application.

If the **Show Login Button** login option was selected, after the IframeContainer is loaded, the **Publish External Login to Iframe** button is displayed. Clicking this button calls the **publishExternalLogin** method to complete the external login, causing the **simpleOrgBrowser** application to be loaded.

See simpleOrgBrowser URL.

- **Load Iframe: Event Viewer** - Clicking this button creates a `com.tibco.wcc.components.IframeContainer` that is set up to load the **simpleEventViewer** WCC application.

  If either the **Use Cached Login** or **Use externalLogin URL Param** login option had been selected, the IframeContainer is directly loaded with the **simpleEventViewer** application.

  If the **Show Login Button** login option was selected, after the IframeContainer is loaded, the **Publish External Login to Iframe** button is displayed. Clicking this button calls the **publishExternalLogin** method to complete the external login, causing the **simpleEventViewer** application to be loaded.

  See simpleEventViewer.

simpleOrgBrowser URL

```
http://wccloginmanagedhubiframesource:9090/wccLoginManagedHub/simpleOrgBrowser.html
```

This is the WCC application that is loaded into the IframeContainer when the **Load Iframe: OrgBrowser** button is clicked. This application contains an **OrganizationResourceBrowser** component (which is a composite component containing the **wccOrganizationBrowser** and **wccOrganizationResourceList** components).

The **postLoadInit** function for this application creates an instance of the **com.tibco.wcc.components.IframeClient**:

```
classProto.postLoadInit = function(path) {
   this.jsxsuper();
   this.iFrameClient = new
com.tibco.wcc.components.IframeClient(this);
};
```

By creating the IframeClient instance, a subscription to the ManagedHub is set up for the externalLogin event. This results in the login being initialized for the application and the **postLogin** method is called, which then renders the **wccOrganizationResourceBrowser** component:

```
classProto.postLogin = function() {
   this.jsxsuper();
   jsx3.GO('wccOrganizationResourceBrowser').render();
};
```

See:

```
...\Samples\wccLoginManagedHub\JSXAPPS\simpleOrgBrowser\application\js
\Application.js
```

simpleEventViewer

```
http://wccloginmanagedhubiframesource:9090/wccLoginManagedHub/simpleEventViewer.html
```

This is the WCC application that is loaded into the IframeContainer when the **Load Iframe: EventViewer** button is clicked. This application contains a **wccEventViews** and a **wccEventViewer** component.

The **postLoadInit** function for this application creates an instance of the **com.tibco.wcc.components.IframeClient**:

```
classProto.postLoadInit = function(path) {
   this.jsxsuper();
   this.iFrameClient = new
com.tibco.wcc.components.IframeClient(this);
};
```

By creating the IframeClient instance, a subscription to the ManagedHub is set up for the **externalLogin** and **externalShowEvents** events. The **externalLogin** results in the login being initialized for the application and the **postLogin** method is called, which then renders the **wccEventViews** component:

```
classProto.postLogin = function() {
   this.jsxsuper();
   jsx3.GO('wccEventViews').render();
};
```

See:

```
...\Samples\wccLoginManagedHub\JSXAPPS\simpleEventViewer\application\js
\Application.js
```

The **publishExternalShowEvents** method publishes the **externalShowEvents** event. This event is handled by the IFrameClient, which calls the **showEvent** Tools Interface method to publish the application local event, which the **EventViewer** component subscribes to by default. When it receives the event, the **EventViewer** component creates a temporary event view, based on the parameters passed in the **publishExternalShowEvents** method, and adds the view to the event view list. (For information about the **showEvent** method, see showEvents

The **publishExternalShowEvents** method is located in `IframeContainer.js` in the following directory:

`*StudioHome*\wcc\*version*\JSXAPPS\components\application\js\`

where:

- *StudioHome* is the directory in which TIBCO Business Studio was installed.
- *version* is the version number of Workspace that was installed with TIBCO Business Studio.

For more information about the **publishExternalShowEvents** method, see Using the TIBCO PageBus Managed Hub with WCC Components.

wccPassThruManagedHub Sample Application

The **wccPassThruManagedHub** sample application illustrates how to use the methods in the **IframeContainer** class.

This sample application illustrates how to do the following when using the PageBus Managed Hub:

- Initialize a login for a WCC application running in an iframe, by utilizing a login from a previous login session.
- Publish a **passThruEvent** event, which passes through the TIBCO PageBus Managed Hub, where the IframeClient subscribes to the event and displays either a work item list or process instance list, depending on the payload in the published event.

The sample is set up to run on a web server at the following URLs:

```
http://wccloginmanagedhubmainsource:9090/wccPassThruManagedHub
http://wccloginmanagedhubiframesource:9090/wccPassThruManagedHub
```

Note that two URL host names are used in this example, specifically to illustrate the TIBCO PageBus Managed Hub iframe setup:

```
http://wccloginmanagedhubmainsource:9090
http://wccloginmanagedhubiframesource:9090
```

This sample application can be run on a single computer running the web server by setting the host names in the computer's `/etc/hosts` file (`C:\WINDOWS\System32\drivers\etc\hosts` on Windows):

- 127.0.0.1 wccloginmanagedhubmainsource
- 127.0.0.1 wccloginmanagedhubiframesource

Since this is a local alias, the web browser must be running on the same computer as the web server for this setup to work.

Setting Up the wccPassThruManagedHub Application

Procedure

1. Add the Workspace framework files to the web server path.

   For example, assuming Tomcat as the web server, copy the following files and directories:

   - `StudioHome\wcc\ver\brand.xml`
   - `StudioHome\wcc\ver\close.html`
   - `StudioHome\wcc\ver\externalform.htm`
   - `StudioHome\wcc\ver\jsx3.gui.window.html`
   - `StudioHome\wcc\ver\logger.xml`
   - `StudioHome\wcc\ver\addins\`
   - `StudioHome\wcc\ver\dojo-toolkit\`
   - `StudioHome\wcc\ver\JSX\`
   - `StudioHome\wcc\ver\JSXAPPS\`

   ... to the following directory:

   ```
   TomcatHome\webapps\wccPassThruManagedHub\
   ```

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *ver* is the version number of Workspace that was installed with TIBCO Business Studio.
   - *TomcatHome* is the Tomcat installation directory.

2. Copy the root HTML files needed by the sample application.

   Copy these files:

   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\IframeContainerTest.html`
   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\simpleLogin.html`
   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\simpleProcessInstancePreview.html`
   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\simpleWorkItemPreview.html`

   ... to the following directory:

   ```
   TomcatHome\webapps\wccPassThruManagedHub\
   ```

3. Copy application directories for the sample application.

   Copy these directories:

   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\JSXAPPS\simpleLogin\`
   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\JSXAPPS\simpleProcessInstancePreview\`
   - `StudioHome\wcc\ver\samples\wccPassThruManagedHub\JSXAPPS\simpleWorkItemPreview\`

   ... to the following directory:

   ```
   TomcatHome\webapps\wccPassThruManagedHub\JSXAPPS\
   ```

4. Configure the **baseUrl** for the Action Processor in the sample application's configuration files:

   1. Open the `config.xml` file in the following directories:

      - *TomcatHome*`\webapps\wccPassThruManagedHub\JSXAPPS\simpleLogin\`
      - *TomcatHome*`\webapps\wccPassThruManagedHub\JSXAPPS\simpleProcessInstancePreview\`
      - *TomcatHome*`\webapps\wccPassThruManagedHub\JSXAPPS\simpleWorkItemPreview\`

   2. Locate the **Action Processor** record and set the **baseUrl** attribute to point to the location of your Action Processor. The string in the **baseUrl** attribute must be in the form:

      ```
      http://Host:Port/bpm/actionprocessor/actionprocessor.servlet
      ```

      where:

      - *Host* is the name or IP address of the machine hosting the BPM runtime.
      - *Port* is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

        The **weighting** attribute is not used at this time.

        For example:

        ```
        <record jsxid="ActionProcessors" type="workspace">
            <ActionProcessor
               weighting="100"
               baseUrl="http://Austin:8080/bpm/actionprocessor/
        actionprocessor.servlet">
            </ActionProcessor>
        </record>
        ```

Executing the wccPassThruManagedHub Application

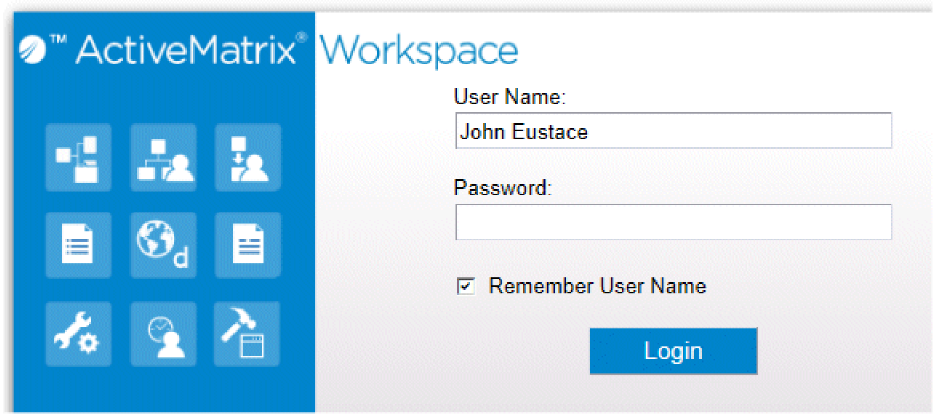
After performing the setup tasks described in Setting Up the wccPassThruManagedHub Application, and starting the web server, you can execute the sample application using the URLs described below.

There are four URLs used in this sample application. They are each described below.

simpleLogin URL

```
http://wccloginmanagedhubmainsource:9090/wccPassThruManagedHub/simpleLogin.html
```

This URL opens the simpleLogin WCC application, which displays the **Login** component:



User Name: John Eustace

Password:

☑ Remember User Name

The **postLogin** function for the simpleLogin sample application has been modified to open another browser window that points to the `IframeContainerTest.html` file, as follows:

```
classProto.postLogin = function() {
   this.jsxsuper();
   var extUrl =    'http://wccloginmanagedhubmainsource:9090/wccPassThruManagedHub/
IframeContainerTest.html';
   var windowRef = window.open(extUrl, 'TestWindow');
};
```

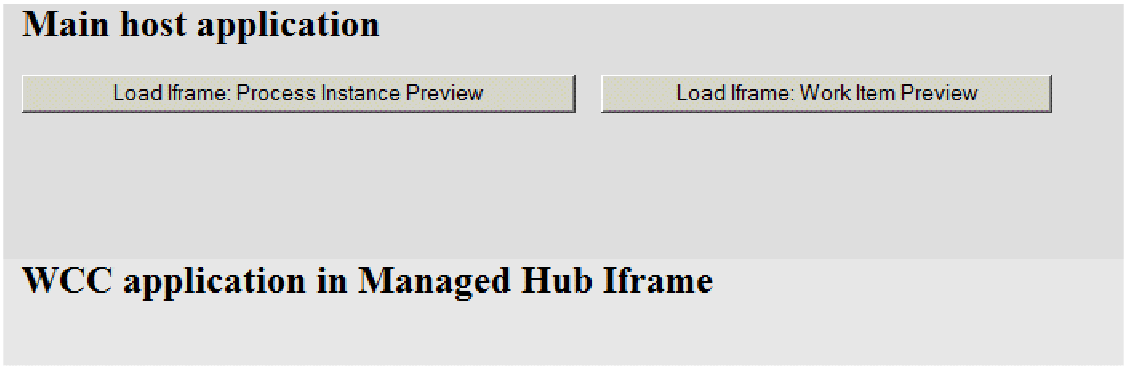
See:

```
...\Samples\wccPassThruManagedHub\JSXAPPS\simpleLogin\application\js\Application.js
```

IframeContainerTest URL

```
http://wccloginmanagedhubmainsource:9090/wccPassThruManagedHub/
IframeContainerTest.html
```

This URL, which is launched from the **postLogin()** function in the simpleLogin application, displays a new browser window that allows you to choose whether to load either a Process Instance Preview component (process instance list with a preview pane) or a Work Item Preview component (work item list with a preview pane) in an iframe:


Main host application
Load Iframe: Process Instance Preview
Load Iframe: Work Item Preview
WCC application in Managed Hub Iframe

The buttons are used to create the iFrameContainer, as follows:

- **Load Iframe: Process Instance Preview** - This button creates a `com.tibco.wcc.components.IframeContainer` that is set up to load the **simpleProcessInstancePreview** WCC application.
- **Load Iframe: Work Item Preview** - Clicking this button creates a `com.tibco.wcc.components.IframeContainer` that is set up to load the **simpleWorkItemPreview** WCC application.

simpleProcessInstancePreview

```
http://wccloginmanagedhubiframesource:9090/wccPassThruManagedHub/
simpleProcessInstancePreview.html
```

This is the WCC application that is loaded into the IframeContainer when the **Load Iframe: Process Instance Preview** button is clicked. This application contains a **Process Instances Preview** component (which is a composite component containing the **Process Instances** component and the preview pane).

The **postLoadInit** function for this application creates an instance of the **com.tibco.wcc.components.IframeClient**:

```
classProto.postLoadInit = function(path) {
   this.jsxsuper();
   this.iFrameClient = new
com.tibco.wcc.components.IframeClient(this);
};
```

By creating the IframeClient instance, a subscription to the PageBus Managed Hub is set up for the externalLogin event. This results in the login being initialized for the application. This application contains a single **Process Instances Preview** component that has been configurated for an external event using the Events Editor in TIBCO General Interface Builder. `IframeContainerTest.html` publishes an event with the configured event payload.

See:

```
...\Samples\wccPassThruManagedHub\IframeContainerTest.html
```

simpleWorkItemPreview

```
http://wccloginmanagedhubiframesource:9090/wccPassThruManagedHub/
simpleWorkItemPreview.html
```

This is the WCC application that is loaded into the IframeContainer when the **Load Iframe: Work Item Preview** button is clicked. This application contains a **Work Items Preview** component (which is a composite component containing the **Work Items** component and the preview pane).

The **postLoadInit** function for this application creates an instance of the **com.tibco.wcc.components.IframeClient**:

```
classProto.postLoadInit = function(path) {
   this.jsxsuper();
   this.iFrameClient = new
com.tibco.wcc.components.IframeClient(this);
};
```

By creating the IframeClient instance, a subscription to the PageBus Managed Hub is set up for the externalLogin event. This results in the login being initialized for the application. This application contains a single **Work Items Preview** component that has been configurated for an external event using the Events Editor in TIBCO General Interface Builder. `IframeContainerTest.html` publishes an event with the configured event payload.

See:

```
...\Samples\wccPassThruManagedHub\IframeContainerTest.html
```

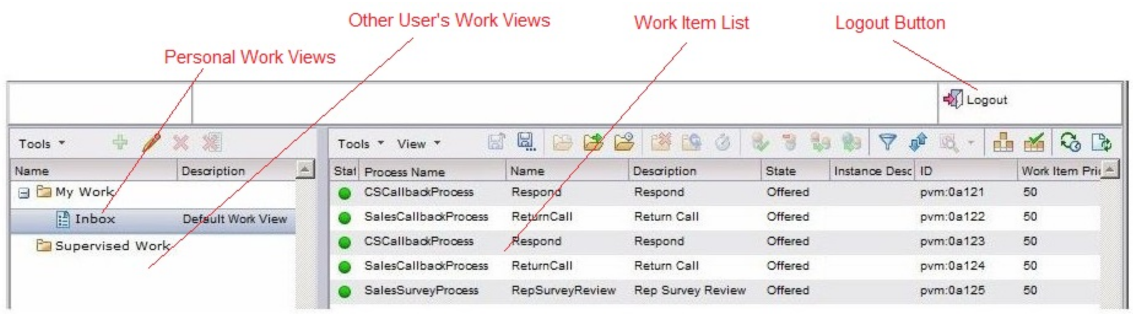
How to Create a BPM Desktop Using Components Tutorial

This tutorial shows you how easy it is to create a fully functioning application using WCC components, without any coding required.

This tutorial steps you through the process of creating an application that does the following:

- Displays the Login dialog and a **Logout** button.
- Displays the **work view list** after the user is authenticated using the Login dialog.
- Displays the **work item list** when the user clicks on a work view in the work view list.
- Shows a “Loading Data” message while the application is retrieving information from the server and rending it on the screen — this is accomplished with the **Data Mask** component.

After logging in, the screen will appear similar to the following:



Name	Description
My Work	
Inbox	Default Work View
Supervised Work	

Stat	Process Name	Name	Description	State	Instance Desc	ID	Work Item Pri
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a121	50
●	SalesCallbackProcess	ReturnCall	Return Call	Offered		pvm:0a122	50
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a123	50
●	SalesCallbackProcess	ReturnCall	Return Call	Offered		pvm:0a124	50
●	SalesSurveyProcess	RepSurveyReview	Rep Survey Review	Offered		pvm:0a125	50

To perform this tutorial, the following software must be installed:

- **TIBCO Business Studio** - This provides the tools needed to create processes and organization models used in TIBCO applications. You must use the version of TIBCO Business Studio that targets the version of TIBCO ActiveMatrix BPM you are using.

  The Business Studio installation also includes the following software, which are used to create and/or customize WCC applications:

  – **TIBCO General Interface** - This consists of two components: General Interface Builder, which is the development environment you will use to develop your custom WCC application, and the General Interface Framework, which is the runtime component for executing your custom WCC application.

  – **Workspace Client Components** - These "WCC" components are used as the building blocks for WCC custom applications. They provide functionality such as the ability to log in, display work item lists, open and process work items, etc. The WCC components are added to custom applications using General Interface Builder.

    When installing TIBCO Business Studio, ensure that the "Workspace Client Components" option is selected.

    For information about installing TIBCO Business Studio, see the *TIBCO Business Studio - BPM Edition - Installation* guide.

- **TIBCO ActiveMatrix BPM Node** - This includes all of the components needed to run TIBCO applications, including custom WCC applications.

  A TIBCO ActiveMatrix BPM Node does not need to be installed prior to developing your custom WCC application, but it must be installed and running before you can run your finished application. For information, see the *TIBCO ActiveMatrix BPM Installation* guide.

Tutorial Procedure

This tutorial describes how to create a fully functioning application using WCC components.

Procedure

1. Start General Interface Builder and create a workspace if you haven't created one yet (the General Interface workspace must be set to the directory that contains the `GI_Builder.html` file, as shown below).

   General Interface Builder can be started by executing the following:

   ```
   StudioHome\wcc\version\GI_Builder.html
   ```

   where:

   - *StudioHome* is the directory in which TIBCO Business Studio was installed.
   - *version* is the version number of Workspace that was installed with TIBCO Business Studio.

     For more information about starting General Interface Builder, see the "Starting TIBCO General Interface Builder" section in *TIBCO General Interface Getting Started.*
2. From the **Project** menu in General Interface Builder, select **New Project**.

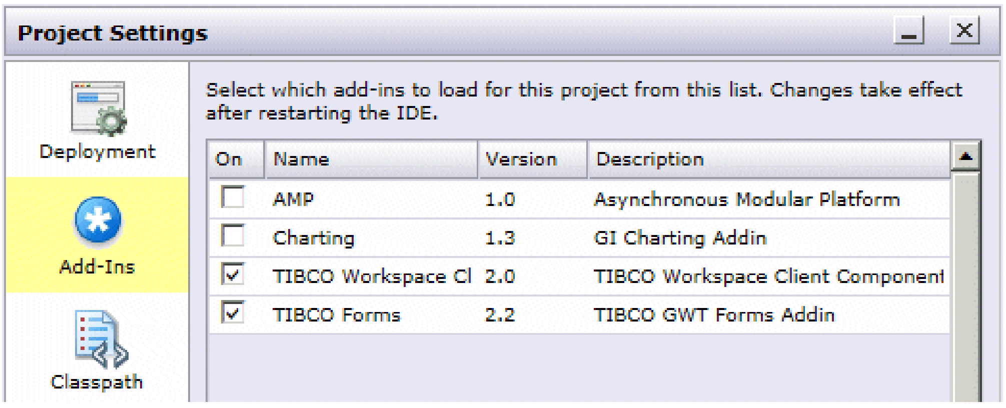
   The Choose a Project Type dialog is displayed.
3. Select a "General Interface Application", then click **Next**.

   The Choose a Project Template dialog is displayed.
4. Select an "Empty Project" template for this tutorial, then click **Next**.

   The Choose a Project Path dialog is displayed.
5. Enter a name for a new project, then click **Finish**. This can be any name because this new project will be closed when you start a "WCC" (Workspace Client Components) project.
6. On the **Project** menu, select **Project Settings**.

   The Project Settings dialog is displayed.
7. Click on the **Add-ins** icon on the left side of the Project Settings dialog.

   Check boxes for the available add-ins are displayed.
8. Click in the **TIBCO Workspace Client Components (WCC)** check box, then click **Save** (leave all other check boxes as you found them — for more information about Add-ins, see the *TIBCO Workspace Configuration and Customization* guide).



Select which add-ins to load for this project from this list. Changes take effect after restarting the IDE.

On	Name	Version	Description
☐	AMP	1.0	Asynchronous Modular Platform
☐	Charting	1.3	GI Charting Addin
☑	TIBCO Workspace Cl	2.0	TIBCO Workspace Client Component
☑	TIBCO Forms	2.2	TIBCO GWT Forms Addin

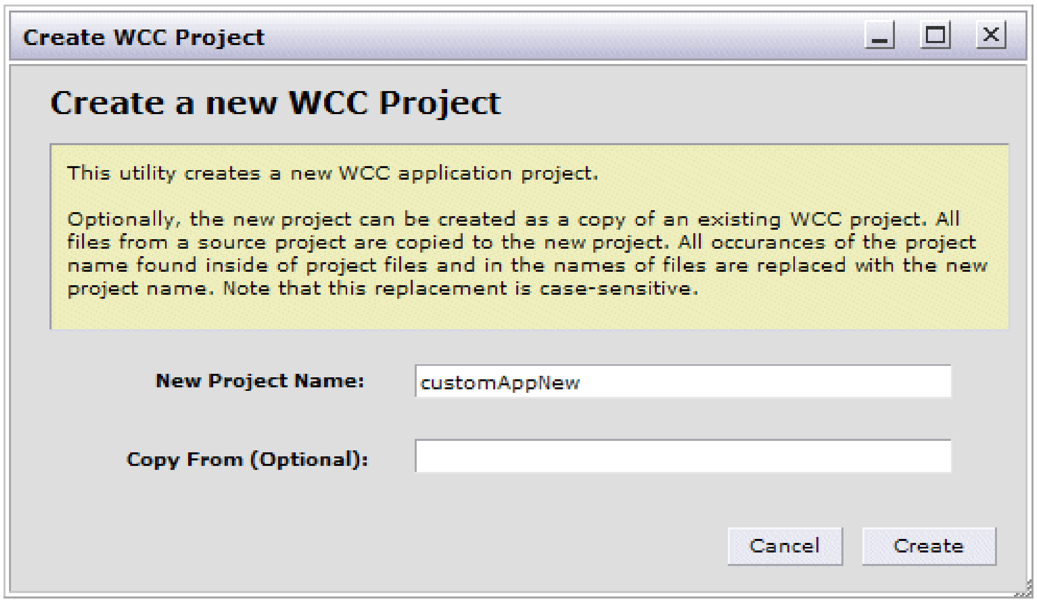
The TIBCO Workspace Client Components add-in provides access to the WCC components from within General Interface Builder at design-time.

A dialog is displayed informing you that you must restart General Interface Builder for the add-in change to take effect. Click **OK** on this dialog.

9. Refresh the browser window to restart General Interface Builder (in Internet Explorer, select **Refresh** from the **View** menu). This causes the TIBCO Workspace Client Components (WCC) add-in to be loaded.
10. From the **Project** menu, select **New Project**.

    The Choose a Project Type dialog is displayed.
11. Choose a "Workspace Client Components (WCC) Application", then click **Next**.

    The following dialog is displayed:



Create WCC Project

Create a new WCC Project

This utility creates a new WCC application project.

Optionally, the new project can be created as a copy of an existing WCC project. All files from a source project are copied to the new project. All occurances of the project name found inside of project files and in the names of files are replaced with the new project name. Note that this replacement is case-sensitive.

New Project Name: customAppNew

Copy From (Optional):

This dialog is used to create a new WCC project. Creating a custom application that includes Workspace components requires that you create a WCC project that is based on an existing WCC project.

A *base-level* WCC project upon which you can base your project is provided. This base-level WCC project includes the Workspace components as an Add-in. This allows you to use those components, as well as the standard General Interface Builder components, to create your custom application.

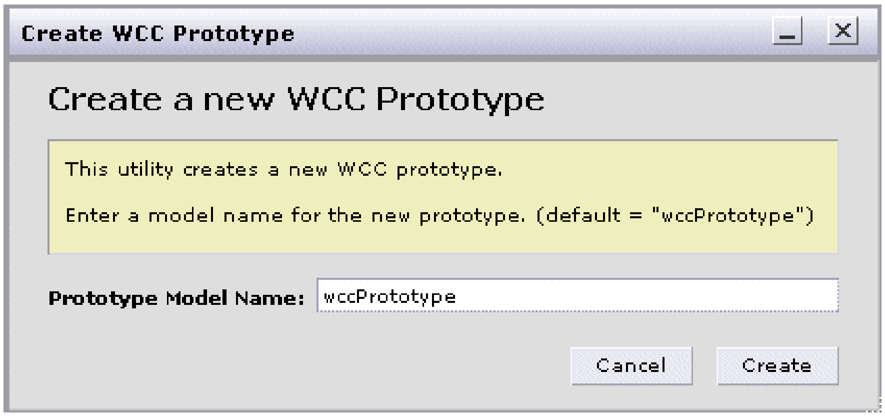
The default is to create your new WCC project based on the provided base-level WCC project. You can override this by specifying, in the **Copy From** field, the name of the existing WCC project upon which you want to base the new project.

12. In the **New Project Name** field, enter the name you would like to give your new project. Use “Accounts” for this tutorial.

    In this tutorial, you’ll use the base-level WCC project as our starting point, and leave the **Copy From** field blank.
13. Click the **Create** button.

    After General Interface Builder reloads the new project, notice the **JSXAPPS/Accounts** link in the lower left-hand corner of the General Interface Builder dialog. This tells you what project you are currently in, as well as provides a way to easily navigate via the browser to the directory that contains all of your project files.
14. From the **New** menu in General Interface Builder, select **GUI Component**.

    The following dialog is displayed:



This utility creates a new WCC prototype.

Enter a model name for the new prototype. (default = "wccPrototype")

Prototype Model Name: wccPrototype

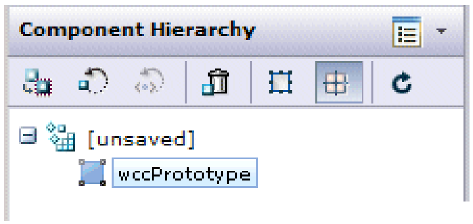
If you are going to add Workspace components to your application, they must be inside of a WCC prototype. You can think of the WCC prototype as a component container.

Creating a WCC prototype sets up the PageBus so that it can wire-up all of the component events you add to the prototype.

15. Enter a *model* name for the prototype, then click **Create**. Use “AccountsPrototype” in this tutorial.

    The prototype model name is used in the *subject* string internally by the PageBus to identify events associated with specific prototypes/components.

    After creating the WCC prototype, the **Component Hierarchy** palette will appear as follows:



Notice that the *model name* does not appear in the Component Hierachy. However, if you view the source XML for the WCC prototype, the model name is defined in the **prototypeModelName** attribute:

```
<strings jsxname="wccPrototype"
   jsxwidth="100%"
   jsxheight="100%"
   appModelName="com.tibco.wcc.Accounts"
   prototypeModelName="AccountsPrototype"
   ...
```

Also, notice that the WCC prototype is actually contained within a top-level prototype object, which is unnamed at this point. This top-level prototype represents a General Interface user-defined prototype in our project — this prototype can contain WCC prototypes.

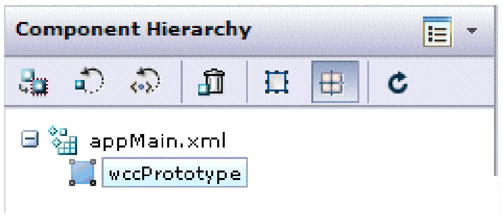
16. From the **File** menu, select **Save** (or right click on the work area tab and select **Save**).

    The file you are saving at this point is the General Interface user-defined prototype. Save the prototype in the `appMain.xml` file, overwriting the existing `appMain.xml` file, in the following directory:

```
   GIWorkspaceDir\JSXAPPS\WCCProjectName\application\prototypes
```

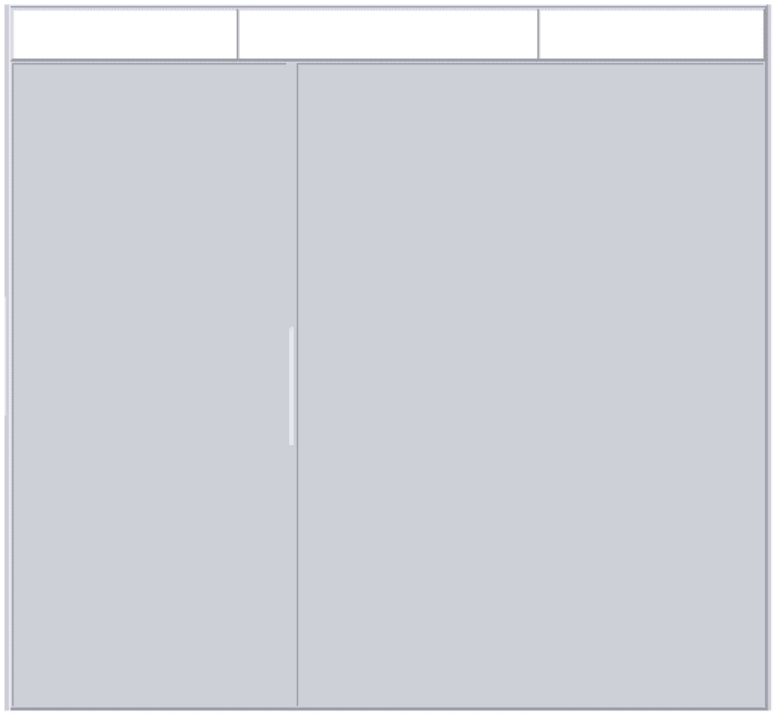
where *GIWorkspaceDir* is the TIBCO General Interface workspace directory you established the first time General Interface Builder was started (see step1) and *WCCProjectName* is the name you gave the WCC Project in step 12 (“Accounts” in this tutorial).

The **Component Hierarchy** palette now appears as follows:



The tab in the work area also now contains the name of the General Interface user-defined prototype, `appMain.xml`.

17. Using the standard TIBCO General Interface Builder Layout and Splitter components, create a screen layout similar to the following:



Note - If you are not familiar with TIBCO General Interface Builder, you can refer to Creating the Tutorial Screen Layout for detailed steps about how to create this layout using the standard General Interface Builder components.

18. From the **Component Libraries** palette, select and drag the WCC **Login** component into the left-side pane created by the vertical splitter.

    Note that you could also drag and drop WCC components onto the appropriate pane in the **Component Hierarchy** palette.

    The Login dialog appears in the pane.

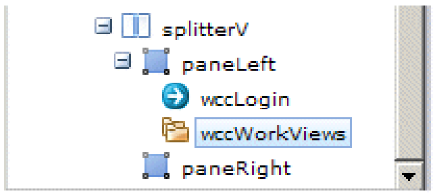
19. From the **Component Libraries** palette, select and drag the WCC **Toolbar Button** component into the top right area created by the Layout components.

    You will later designate this toolbar button the **Logout** button.

20. From the **Component Libraries** palette, select and drag the WCC **Data Mask** component into the top center area created by the Layout components.

The **Data Mask** component provides a “Loading Data ...” message while the application is retrieving data from the server and rendering it on the screen.

21. From the **Component Libraries** palette, select and drag the WCC **Work Views** component onto the appropriate pane in the **Component Hierarchy** palette that represents the area to the left of the vertical splitter, i.e., the pane in which the **Login** component is located.



Note that you cannot place the **Work Views** component directly in the work area because the **Login** component is already in that area. (When a component is dropped into a container in the work area, it, by default, consumes 100% of the container. If you want to place another component in the same container, General Interface Builder requires that you place it in the pane in the Component Hierarchy tree.)

The **Work Views** component displays a list of work views (note that by default, the only work view is the "Inbox" — you can create additional work views using the view wizard, if desired). You will later configure events so that this list is displayed when the user is authenticated through the **Login** component.

When dragging and dropping WCC components onto the **Component Hierarchy**, do not drop them anywhere above the WCC prototype in the component tree. WCC components must be placed in a WCC prototype.

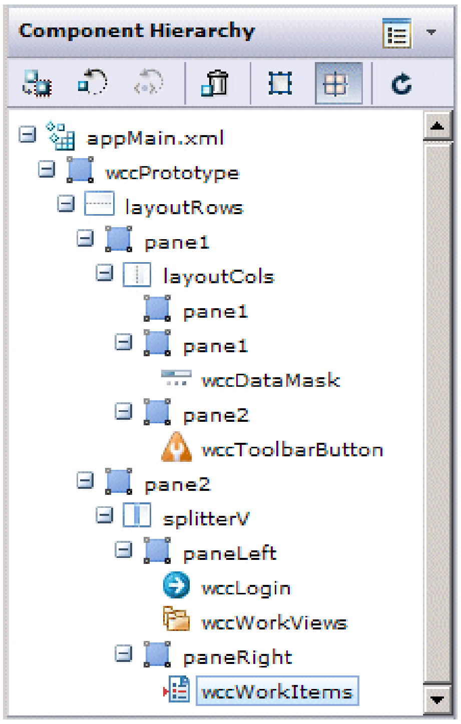
22. From the **Component Libraries** palette, select and drag the WCC **Work Items** component into the right-side pane created by the vertical splitter.

The **Work Items** component displays a work item list containing the work items for a particular work view (the user’s "Inbox", by default, contains all of that user’s work items).

You will later configure events so that this list is displayed when the user clicks on a view in the work view list.

Notice that a grid pattern representing each component is shown in the containers to provide visual feedback.

The **Component Hierarchy** should now look like this (if the components are not in this order, move them by dragging and dropping them in the proper location):

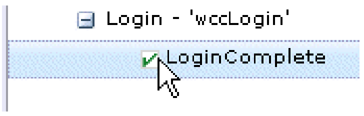


Now you'll wire-up the events that will cause the work view list to display when the user is authenticated, and the work item list to display the work items in the selected work view.

23. In the **Component Hierarchy** palette, click on the **wccWorkViews** component, then click on the **wccWorkViews** button in the taskbar (lower left part of the General Interface Builder dialog) to display the Properties/Events Editor dialog.
24. Click on the **Events** tab.

> In this tutorial, you will not change any of the property settings on the **Properties** tab for the purpose of specifying access to the functions provided by the component. For information about setting properties, see Properties Editor .
>
> You will, however, set a property on the **Toolbar Button** component to designate it as a **Logout** button — this is explained later.

25. Double-click the **LoginComplete** event, which causes the check box to become checked:



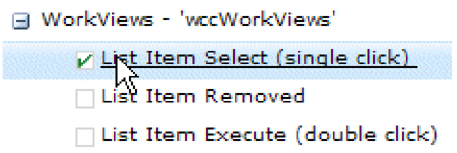
This causes the **wccWorkViews** component to subscribe to the "LoginComplete" event, which the **Login** component publishes.

This results in the work view list being rendered when the user's login credentials are authenticated.

26. Click **Commit** to save the changes.
27. In the **Component Hierarchy** palette, click on the **wccWorkItems** component.

This causes the **Properties/Events Editor** to now control the properties and events for the **wccWorkItems** component — the header bar on the Properties/Events Editor dialog displays "wccWorkItems".

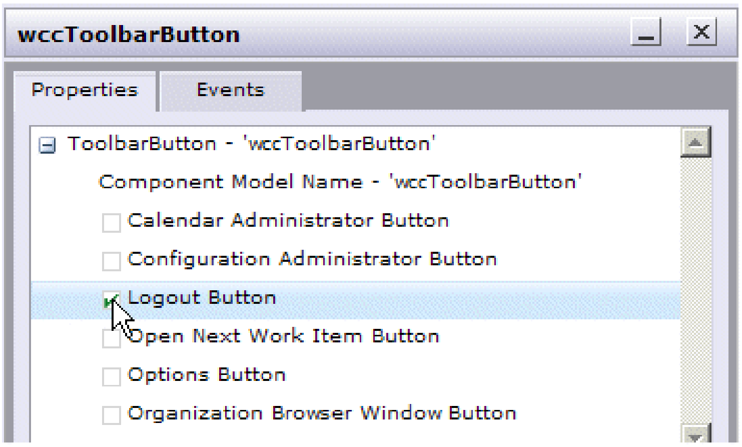
28. On the Properties/Events Editor dialog **Events** tab, double-click on the "List Item Select (single click)" event for the **wccWorkViews** component.


WorkViews - 'wccWorkViews'
List Item Select (single click)
List Item Removed
List Item Execute (double click)

This causes the **wccWorkItems** component to subscribe to the “List Item Select (single click)” event that the **wccWorkViews** component publishes.

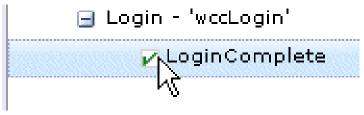
This results in the work item list being rendered when a work view in the work view list is selected (note that the first work view in the list is selected by default when the work view list is displayed, causing that work item list to be automatically displayed when the work view list is rendered).

29. Click **Commit** to save the changes.
30. In the **Component Hierarchy** palette, click on the **wccToolbarButton** component.
31. On the Properties/Events Editor dialog **Properties** tab**,** double-click the “Logout Button” property:


wccToolbarButton
Properties
Events
ToolbarButton - 'wccToolbarButton'
Component Model Name - 'wccToolbarButton'
Calendar Administrator Button
Configuration Administrator Button
Logout Button
Open Next Work Item Button
Options Button
Organization Browser Window Button

This causes the **wccToolbarButton** component to render as a **Logout** button.

32. While the **wccToolbarButton** component is still selected, click on the Properties/Events Editor dialog **Events** tab**,** then double-click the “LoginComplete” event:



This causes the **wccToolbarButton** component to subscribe to the “LoginComplete” event that the **wccLogin** component publishes.

This results in the **Logout** button being rendered when the user’s login credentials are authenticated.

33. Click **Commit** to save the changes, then minimize or click **Close** to close the editor dialog.
34. From the **File** menu, select **Save**.

You have now created an application that will allow a user to log in and view their work views and display a list of work items for each view. They can also perform any functions available from these lists, for which they have the proper permissions.

The next three steps describe how to implement a change to make the browser window close when the user clicks the **Logout** button.

As an alternative, you can make the Login dialog be re-displayed when the user logs out by reloading the **appMain** prototype (`appMain.xml`). To do this, perform the steps in Reloading the appMain Prototype Upon Logout, then proceed to step 38.

35. Open the `Application.js` file in the following directory:

    ```
    GIWorkspaceDir\JSXAPPS\WCCProjectName\application\js
    ```

    where *GIWorkspaceDir* is the TIBCO General Interface workspace directory and *WCCProjectName* is the name you gave the WCC Project (“Accounts” in this tutorial).

36. Locate the **postLogout** function and add the following line:

    ```
    window.close();
    ```

    This will cause the browser window to close when the user clicks the **Logout** button.

    For more information about using the **postLogout** function, see postLogout.

37. Save and close the `Application.js` file.

38. Open your custom application’s `config.xml` file, which is located as follows:

    ```
    GIWorkspaceDir\JSXAPPS\WCCProjectName\config.xml
    ```

    where *GIWorkspaceDir* is the TIBCO General Interface workspace and *WCCProjectName* is the name you gave the WCC Project in step 12 (“Accounts” in this tutorial).

39. Locate the **ActionProcessors** record and set its **baseUrl** attribute to the location of the Action Processor. The string in the **baseUrl** attribute must be in the form:

    ```
    http://Host:Port/bpm/actionprocessor/actionprocessor.servlet
    ```

    where:

    - `Host` is the name or IP address of the machine hosting the Action Processor. This must be the same machine on which the application is running (i.e., the Action Processor and the application must be running on the same machine).
    - `Port` is the port number used by the ActiveMatrix WebApp Implementation Type to communicate with web applications.

    In a production environment, the **baseUrl** is normally set to an empty string, which causes the URL of the Action Processor to be determined from the URL entered in a browser by the user to start the application. In a testing environment, however, you will be starting the application via the file system, therefore the Action Processor URL needs to be explicitly specified in the **baseUrl** attribute.

40. Save and close the `config.xml` file.

41. In Windows Explorer, navigate to the General Interface workspace directory and execute the *ProjectName*`.html` file. In this tutorial, that is `Accounts.html`. (Note that you can also test/run the application from within General Interface Builder, rather than via Windows Explorer.)

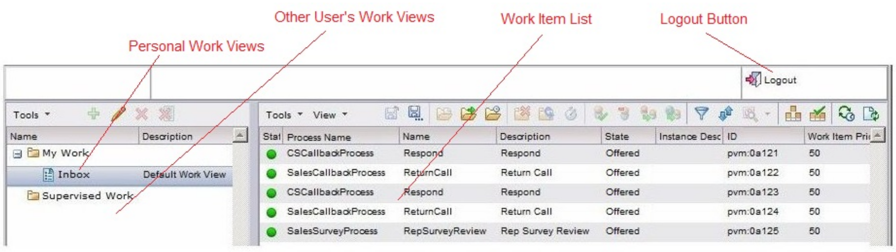
    The Login dialog is displayed.

42. Enter a valid user name and password, then click **OK.**

    By default, Workspace is configured with a default user that can log in so that other users can be configured using the Organization Browser component (for information about configuring users, see the *TIBCO Organization Browser User’s Guide*, or the *How to Map Users to the WelcomeUsers Organization Model* tutorial).

    The default user is "tibco-admin" with a password of "secret". If additional users have been configured on your system, this admin user name and password may have been changed. If so, contact your system administrator for a valid user name and password.

    The default work view (Inbox) is displayed in the left pane, and the work item list is displayed in the right pane. For example:



Name	Description
My Work	
Inbox	Default Work View
Supervised Work	

Stat	Process Name	Name	Description	State	Instance Desc	ID	Work Item Pri
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a121	50
●	SalesCallbackProcess	ReturnCall	Return Call	Offered		pvm:0a122	50
●	CSCallbackProcess	Respond	Respond	Offered		pvm:0a123	50
●	SalesCallbackProcess	ReturnCall	Return Call	Offered		pvm:0a124	50
●	SalesSurveyProcess	RepSurveyReview	Rep Survey Review	Offered		pvm:0a125	50

Note that if you log in as the tibco-admin user, there will not be any work items listed in the work item list as work items do not get sent to the tibco-admin user.

43. Perform any of the available functions for which you have permission.

    Clicking on the **Logout** button logs you out of the server, then either closes the browser window or re-displays the Login dialog, depending on which steps you performed earlier.

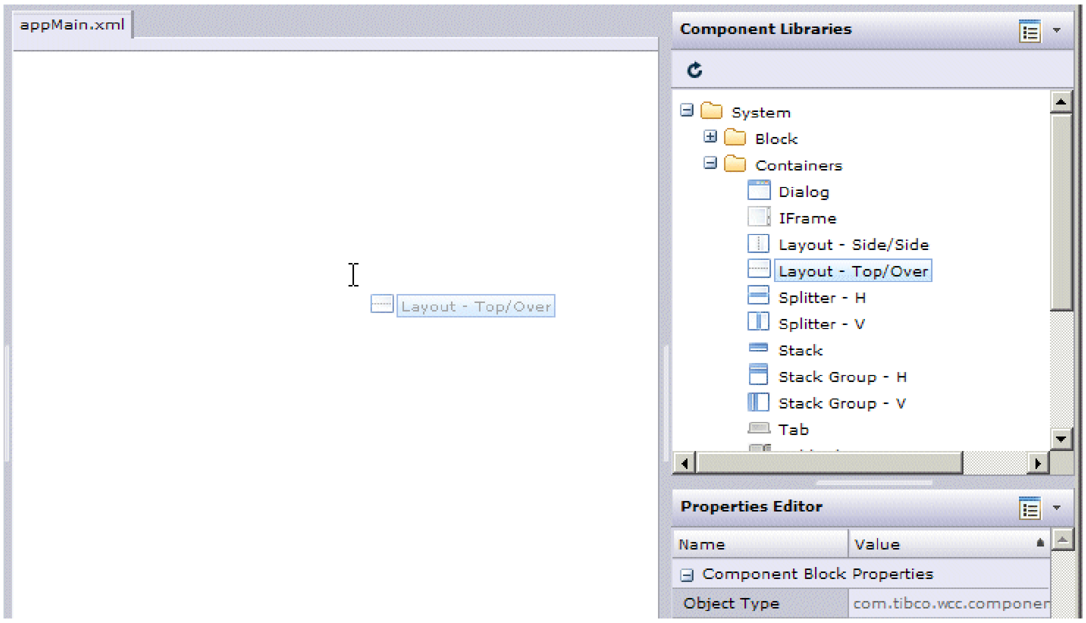
Creating the Tutorial Screen Layout

This procedure shows you how to created the screen layout needed in the How to Create a BPM Desktop Using Components tutorial.

In step 17 of the How to Create a BPM Desktop Using Components tutorial, you are asked to create a layout using the standard General Interface Builder components. If you are not familiar with TIBCO General Interface Builder, you can use the following procedure to help you create the layout.

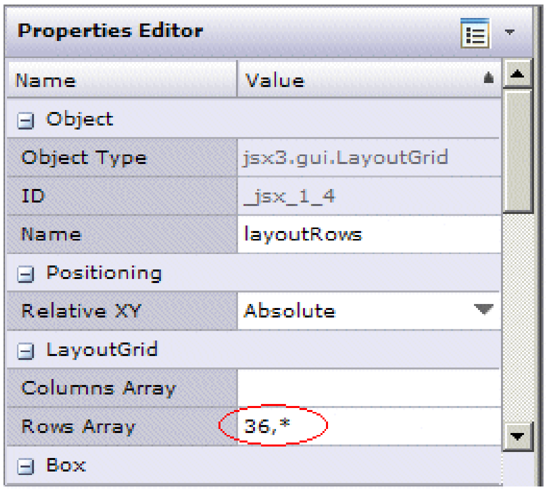
Procedure

1. From the **Component Libraries** palette, expand **Containers** and drag a **Layout - Top/Over** component into the `appMain.xml` canvas.



Name	Value
Component Block Properties	
Object Type	com.tibco.wcc.componer

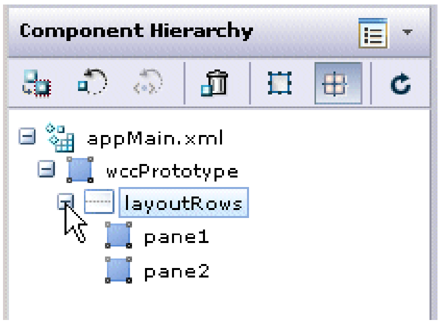
2. In the **Properties Editor** palette in the lower right part of the screen, change the **Rows Array** property value to “**36,***”.



Properties Editor

Name	Value
⊟ Object	
Object Type	jsx3.gui.LayoutGrid
ID	_jsx_1_4
Name	layoutRows
⊟ Positioning	
Relative XY	Absolute
⊟ LayoutGrid	
Columns Array	
Rows Array	36,*
⊟ Box	

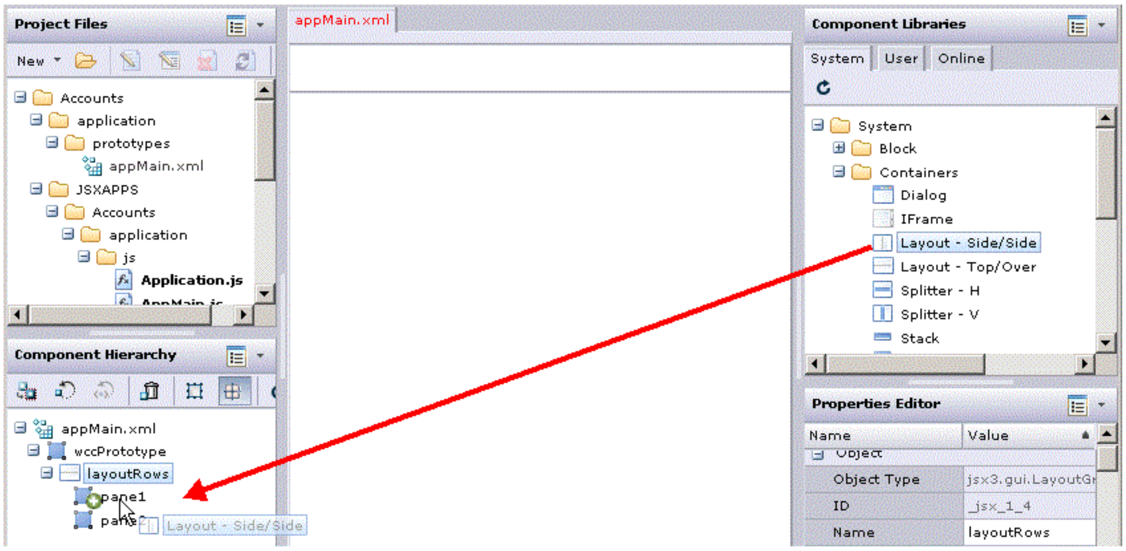
3. Expand the **layoutRows** component in the **Component Hierarchy** palette.



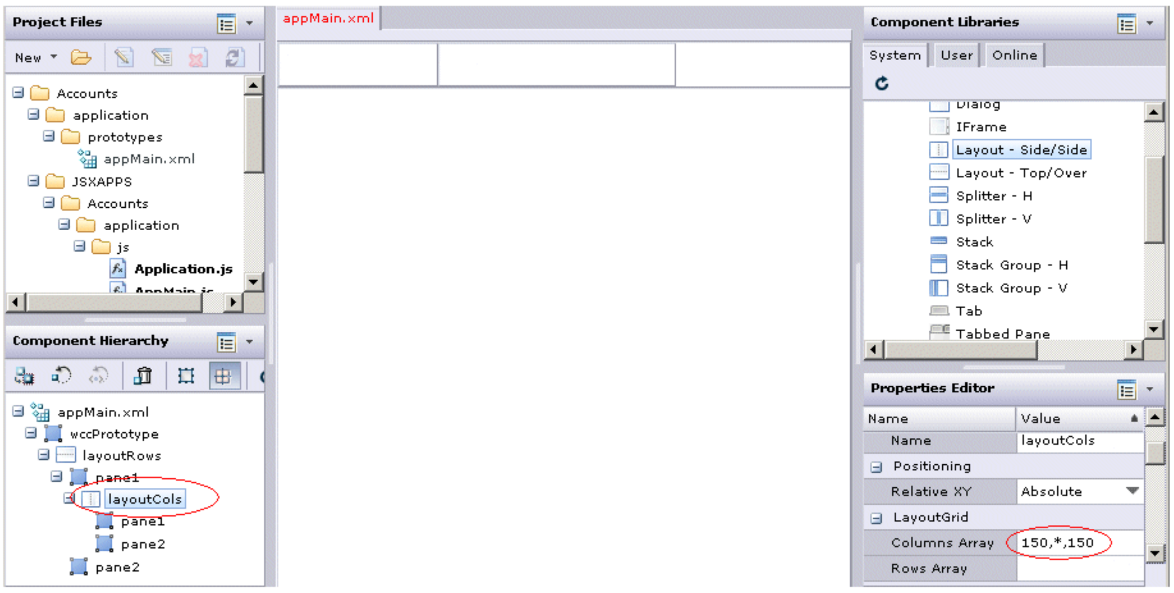
Component Hierarchy

```
⊟ appMain.xml
  ⊟ wccPrototype
    ⊟ layoutRows
        pane1
        pane2
```

4. From the **Component Libraries** palette, drag a **Layout - Side/Side** component into the top pane of the **layoutRows** component in the **Component Hierarchy** palette.

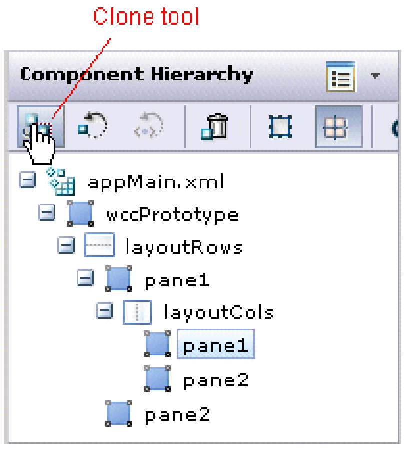


5. Expand and then select the **layoutCols** component in the **Component Hierarchy** palette, then in the **Properties Editor** palette, change the **Columns Array** property value to “**150,*,150**”.

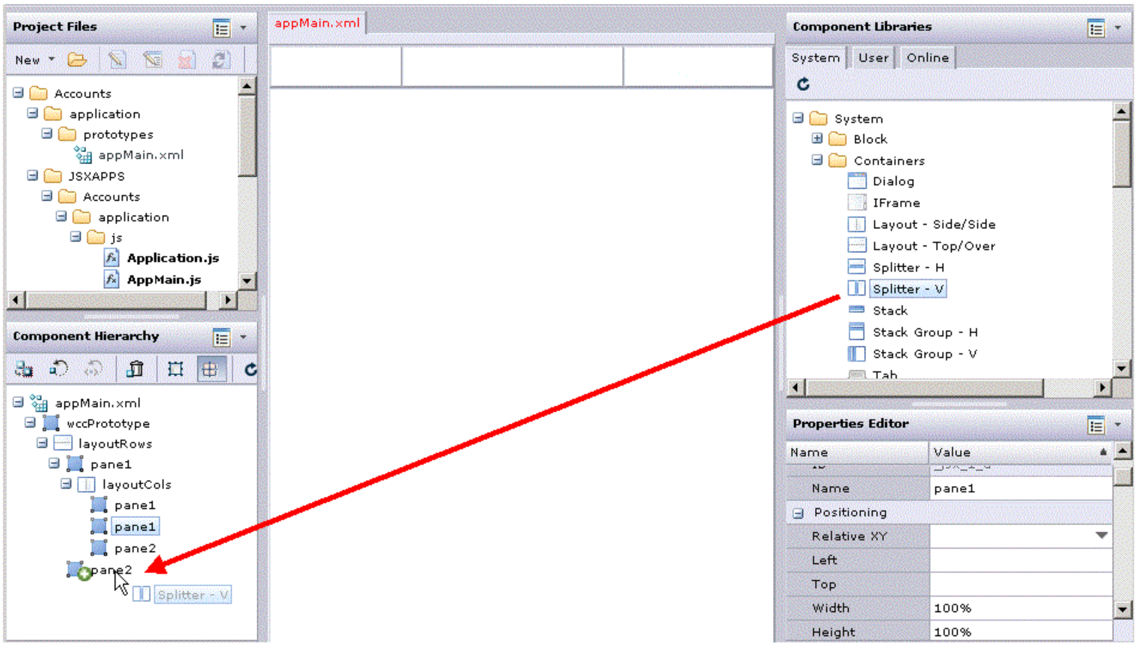


Name	Value
Name	layoutCols
Positioning	
Relative XY	Absolute
LayoutGrid	
Columns Array	150,*,150
Rows Array	

6. In the **Component Hierarchy** palette, select the top **pane** component, under **layoutCols** and then click the **Clone** tool.

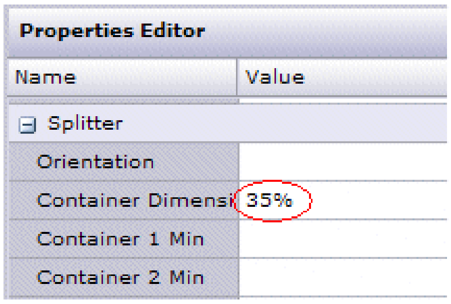


7. From the **Component Libraries** palette, drag a **Splitter - V** component into the bottom **pane** of the **layoutRows** component in the **Component Hierarchy** palette.



Name	Value
Name	pane1
Positioning	
Relative XY	
Left	
Top	
Width	100%
Height	100%

8. Select the **splitterV** component in the **Component Hierarchy**, then in the **Properties Editor**, change the **Container Dimension** from 100% to 35%:



Name	Value
Splitter	
Orientation	
Container Dimensi	35%
Container 1 Min	
Container 2 Min	

9. Save the new layout you've just created.

Result

You can now proceed with step 18 of the tutorial.

Reloading the appMain Prototype Upon Logout

This procedure is used conjunction with the How to Create a BPM Desktop Using Components tutorial; it shows you how to re-display the Login dialog when the user clicks **Logout.**

In step 35 of the tutorial, you are instructed to add `window.close()` to the **postLogout** method, which causes the browser window to close when the user clicks the **Logout** button.

This procedure describes an alternative to closing the browser window upon logout. Performing this procedure causes the **appMain** prototype (`appMain.xml`) to be reloaded when the user clicks **Logout**. This causes the Login dialog to be re-displayed.

Procedure

1. In TIBCO General Interface Builder **Component Hierarchy**, click on **wccPrototype**.

2. Click on the **wccPrototype** button in the task bar, which displays the Properties/Events Editor.
3. In the field on the bottom of the Properties tab, change "wccPrototype" to "AccountsPrototype", click the **Update** button, then click **Commit**.
4. Open the `AppMain.js` file in the following directory:

   ```
   GIWorkspaceDir\JSXAPPS\WCCProjectName\application\js
   ```

   where *GIWorkspaceDir* is the TIBCO General Interface workspace directory and *WCCProjectName* is the name you gave the WCC Project (“Accounts” in this tutorial).

   The `AppMain.js` file is an application-level class mixin interface that contains an example custom post logout method (**customPostLogoutExample**). This custom post logout method will be used to reload the application upon a logout. (The **customPostLogoutExample** method is called from the **postLogout** method in `Application.js`.)
5. Locate the **customPostLogoutExample** method in `AppMain.js` and change the name of the referenced prototype (’sampleAppPrototype’) to "AccountsProtoType". (There are five references to ’sampleAppPrototype’.)
6. Save and close the `AppMain.js` file.
7. Proceed to step 38.

Result

For more information about the **customPostLogoutExample** method, see customPostLogoutExample.