Process Director Documentation

Developer's Reference Guide

Last Updated: 2025-05-09, 14:17

Contents

Documentation Formatting Note

Text and Code Formatting Conventions

To highlight terms and concepts that have special relevance, this documentation implements several formatting conventions to make key words and terms more noticeable.

- Control Label: This format will identify the text labels or properties for Process Director objects, or the names of dialog boxes.
  Example: The Name text box.
- UI Element: This format will identify user interface elements such as buttons, tabs, or other UI objects used to perform interface operations.
  Example: The Submit button.
- Formal Control Name: This format will identify named Process Director controls.
  Example: A Section End control.
- Process Director Object: This format will identify named instances of Process Director Folders, Forms, Process Timelines, Knowledge Views, etc.
  Example: The Travel Expense Approval Process Timeline.
- **Key Terms**: This format will identify key terms and concepts introduced into the text of the document, and which are important to learn.
  Example: A **Case** is group of processes, transactions, or responses that define a complex activity.
- `Code:` This format will identify code samples, system variables, formulas, or other fixed programmatic syntax.
  Example: Type the following formula: `AirFare + Lodging`.
- `Code Option:` A section of a code sample to denote placeholder values that must be replaced by the user manually at design time.
  Example: `{CURR_USER, format=FormatType}`
- `Code Comment:` A section of a code sample that is used for text comments, rather than runnable code.
  Example: `// This is a comment.`
- `Code Variable`: A programming object whose value is usually determined from a command written in code.
  Example: `var formControls = BaseCurrentForm.FormControls;`

In addition to the above, extended samples of program code are presented in a special format to set them off from the rest of the text, as demonstrated below:

```
// Called after database initialized
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Before we make SDK calls that access the database,
    // ensure DB has been opened
    if (bp.DBOpenComplete)
    {
        // Place custom code here
    }
}
```

Important text or warnings are presented in a special callout box for special attention:

> ⊘ **This is an Important item.**

Notes of general interest are also presented in special callout boxes:

> ⓘ **This is a note.**

Hopefully, the use of these formatting conventions will make it easier for you to determine the various types of objects to which the text refers.

Icons

Some universal icons are used in the documentation. They are listed below:

ICON	NAME	DESCRIPTION
#	Link	A hyperlink to the specific URL and named anchor of a topic, heading, or other item.
+	Dropdown Closed	An icon that, when clicked, will expand dropdown text in a topic.
−	Dropdown Open	An icon that, when clicked, will close the expanded dropdown text in a topic.

Finally, some topic headers within each online document may display a link symbol (#) when you mouse over the header. Clicking the link will navigate to that specific section of the document, which can then be bookmarked in your browser.

Other Conventions

URLs displayed in sample will, unless used for commands or URLs used on the local host machine, use the "HTTPS" prefix by default, as modern practice has evolved to use the encryption layer to access URLs, instead of the plain-text method (HTTP) of accessing URLs.

Development Overview

This document describes the code-based customization that is available for Process Director, with the appropriate SDK license. This guide is intended for companies that require customization of process or form processing, or that require integration into other applications.

Script Types

Process Director enables you to create Knowledge View, Form, and Process Timeline script types. All script files are stored as ASCX files in Process Director. They can be placed anywhere in the Content List and are referenced (pointed to) by the object definitions that call them.

.NET Forms

The .NET Form enables you to develop forms completely inside the Visual Studio environment to enable the full functionality that .NET offers.

Customization through Scripting

Process Director provides customization options and APIs that allow the product to provide specialized business logic needed by your organization. The programming API enables you to customize the various functions and interfaces of Process Director; it also enables the product to interface with external systems via API calls made through scripting. There are three main areas where custom scripts are implemented: Form processing, Timeline processing, and configuring user options. This customization isn't required, but is important when you want to perform specific business logic for your requirements. The customization is provided by allowing you to create and write custom .NET controls and functions. These custom functions can get and/or modify data within Process Director or external applications.

To develop Scripts inside Visual Studio, use the fully functional Visual Studio project installed with the product named bpVS.zip. This project includes the DLL's necessary to use Intellisense and compile-time error checking.

Additionally, you can run Process Director inside Visual Studio. This will enable setting of breakpoints in custom scripts, inspection of objects, viewing logs in the

Visual Studio output window, and other debugging techniques. To accomplish this, license and install Process Director on a test/development workstation or server. Then launch Visual Studio, select File->Open Web Site, and select the website folder where Process Director is installed (typically C:\Program Files\BP Logix\Process Director\website). If prompted, don't upgrade to .NET 4. Select Debug-Start Debugging. If prompted, allow Visual Studio to modify web.config to enable debugging.

If you are running Process Director on a development server, you can also edit the web.config manually to enable debugging, by adding the attribute debug="true" in the <compilation> section, as shown below. This is NOT a recommended practice for any production system!

```
<configuration>
    <system.web>
        <compilation debug="true" targetFramework="X.X"/>
    </system.web>
</configuration>
```

> 🛑 **If you manually edit the web.config file, you should be aware that Process Director will overwrite the web.config file on every full or patch installation of Process Director.**

To set breakpoints, open the relevant forms or scripts from the FormCache or WfScriptCache folder. Once opened, breakpoints can be set, and variables can subsequently be inspected or modified. Notice that the scripts and forms are overwritten in the cache folders if they are changed. You may need to right‑click Refresh the cache folders in Visual Studio to see new files appear.

Custom Scripting/Development

Process Director offers a number of methods for creating custom scripts. Script handlers with the appropriate stubs for scripting events are provided for .NET developers in the Visual Studio IDE, or created automatically in the Process Director interface. The term Custom Script can mean a wide variety of things in Process Director. Indeed, a better term might be "Custom Development" since "scripting" can range from a small script to change the background colors of a Knowledge View row, to building an fully functional Process Director application in ASP.NET—though the latter would be an exceptionally rare use case.

This section of the documentation covers the various scripting/development use cases, and provides instructions for implementing several custom development scenarios. You can navigate to each of the topics in the section via the Table of Contents displayed on the upper right corner of the page, or by using the links below.

Installing the BP Logix Visual Studio Plugin: Instructions for setting up Visual Studio with the BP Logix Plugin.

Form Scripts: Creating custom scripts that run for Form instances.

Process Scripts: Creating custom scripts that run for Process Timeline instances.

Knowledge View Scripts: Creating custom scripts that run for Knowledge Views.

Custom ASPX Pages: Creating ASPX pages to run custom scripts, such as for implementing custom business logic, independent of other Process Director objects.

Creating ASP.NET Forms: Creating custom ASCX Forms for use when developing in the Visual Studio environment.

Custom Portlets: Creating custom Workspace portlets to display in the product UI.

Custom Tasks: Creating your own Custom Tasks for use in your applications.

SDK Classes: Reference documentation for the Properties, Methods and Events of all .NET classes contained in the Process Director SDK.

Installing the BP Logix Visual Studio Plugin

The BP Logix Visual Studio Plug-in integrates with Microsoft Visual Studio. It provides the ability to drag and drop Process Director Controls onto your forms. The easiest way to take advantage of the BP Logix Visual Studio Plug-in, use the

fully functional Visual Studio project installed with the product named bpVS.zip. This project already has the plug-in installed and Intellisense enabled.

The following is a list of the features that are available to you in the plug-in.

Drag and Drop Control Editing. The plug-in adds an extension to the Visual Studio Toolbox giving you a list of all controls that are available to you in the Process Director library.

- Control Properties. Properties can now be set in the Properties box of the selected control.
- Intellisense. All controls support full Intellisense to make the parameters easier to configure while in the Source View.
- Compile. Visual Studio will highlight compile errors.

To enable this plug-in click on the menu item at the top named View > Toolbox. Once the toolbox is open right click in an open area and select "Add Tab". Name the tab "Process Director". Ensure the new tab is selected and right-click on the tab and select Choose Items from the list.
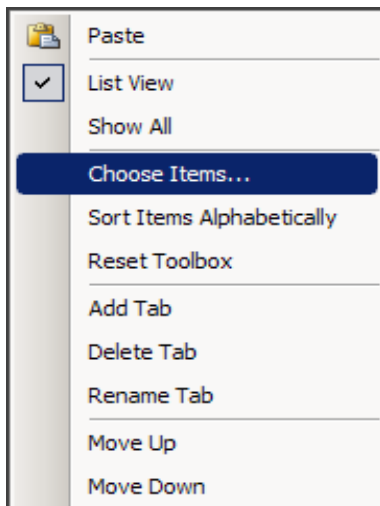
A dialog box will display, select the Browse button under the .Net Framework Components tab. The browse dialog box will appear. If you have Process Director installed, you'll find the DLL at "C:\Program Files\BP Logix\Process Director\". If you only have the plugin installed, you can find the DLL at "C:\Program Files\BP Logix\Plugin\bpVS\Bin". Once you've navigated to that folder, select "BPLogix.bpVSPlugin.dll"

Then click Open and OK.

> ⛔ **Some legacy elements in Process Director, which remain in the product for backwards compatibility, have a dependence on the .NET Framework v3.5, so you must install the v3.5 framework for the plug-in to work as expected. To add this .NET Framework version, click the *Start* button, click *Control Panel*, click *Programs*, and then click *Turn Windows features on or off*. Select *.NET Framework 3.5* from the list of Windows features, then click the *OK* button to install it.**

Restart your Visual Studio application for the complete changes to take effect.
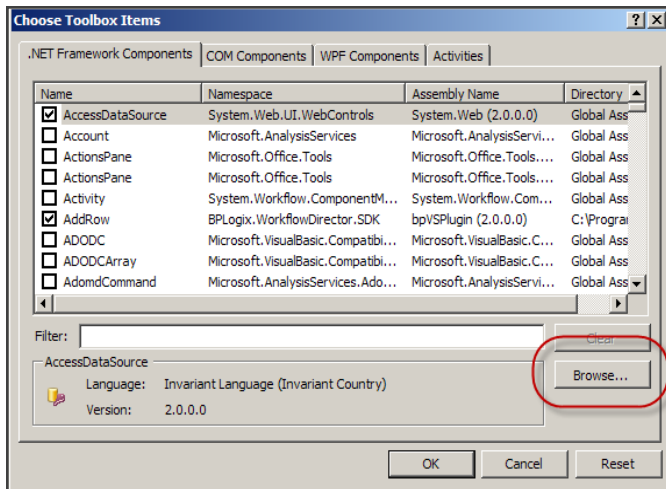
After Visual Studio restarts, the plugin will populate the controls in the tab you created. You can now drag and drop the controls on the form in design mode or split mode only. You'll also be able to use the BP Logix Intellisense. By typing your code you'll start to see Process Director Controls, classes, etc. show in the list provided from the Intellisense.

To enable the Intellisense inside Visual Studio, you should create a stand-alone ASP.NET Web Application, and then add the BPLogix.bpVSPlugin.dll to the Bin directory of the project. To create new Forms or scripts, use Add New Item->Web User Control to create a new .ascx file. Do not place code in separate file. For Forms, you can switch into Design or Split view to drag and drop controls onto the visual form. You made need to build the application to enable the Intellisense. Ensure all Forms and script files have this line at the top of the file:

```
<%@ Control Language="C#" AutoEventWireup="false"
    Inherits="BPLogix.WorkflowDirector.SDK.bpFormASCX" %>
```
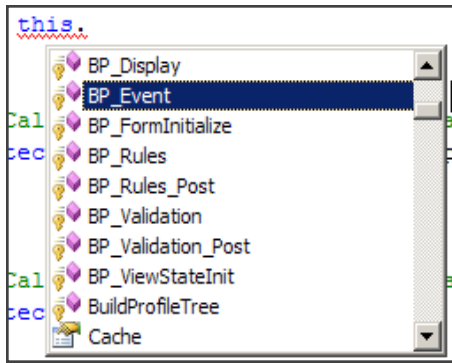
Additionally, you can add the following registry key to set the location where the Plugin downloads temporary files (such as scripts and Forms) by setting the DocFolder property:

`HKEY_LOCAL_MACHINE\SOFTWARE\BP Logix\Plugin\DocFolder`

Or on a 64-bit operating system:

`HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\BP Logix\Plugin\DocFolder`

If you set this to a folder that is in your stand-alone Web Application, then you can use Check Out and Edit for Forms and scripts. When the files are opened in Visual Studio, they'll gain the benefit of Intellisense and compile-time error checking.

Form Scripts

 Form scripts can be used for both Forms created with the Online Form Designer and custom forms created as .ASCX files.

To develop Form Scripts inside Visual Studio, use the fully functional Visual Studio project installed with the product, which is bundled into a ZIP archive named bpVS.zip. This ZIP file is located in the c:\Program Files\BP Logix\Process Director\ directory by default. You can copy this utility to any computer that runs Visual Studio 2013 or higher. Cloud customers can download the plugin from the Downloads section of the BP Logix support site.

Refer to the sample files eform_*.ascx, as well as the SamplePage.aspx file referenced in the Custom ASPX Pages section of the documentation, below.

Scripts for Forms created with the Online Form Designer should be placed into a separate .ASCX document in the Content List. Here is the structure for this type of script:

```
<%@ Control Language="C#" AutoEventWireup="false"
    Inherits="BPLogix.WorkflowDirector.SDK.bpFormASCX" %>
```

```
<script runat="server">
    // Events...
</script>
<script>
    // Insert client-side JavaScript code and functions
here...
</script>
```

Scripts for .ASCX Forms are typically placed in the same file as the .ASCX source form. Here is the structure for this type of form:

```
<%@ Control Language="C#" AutoEventWireup="false"
    Inherits="BPLogix.WorkflowDirector.SDK.bpFormASCX" %>
    // Actual Form contents...
<script runat="server">
    // Events...
</script>
<script>
    // Insert client-side JavaScript code and functions
here...
</script>
```

Form Script Events

In both cases, the APIs and events available are the same. All events will be called with the following environment:

LOCAL VARIABLE	DESCRIPTION
CurrentForm	Reference to the current Form instance object
CurrentUser	Optional instance of the current User object
CurrentPartition	Instance of the current Partition object
CurrentWorkflow	Optional instance of the current Workflow object(Deprecated)
CurrentWorkflowStep	Optional instance of the current WorkflowStep object(Deprecated)
CurrentWorkflowStepUser	Optional instance of the current WorkflowStepUser object (Deprec-

LOCAL VARIABLE	DESCRIPTION
	ated)
CurrentProject	Optional instance of the current Project object
CurrentProjectActivity	Optional instance of the current ProjectActivity object
CurrentProjectActivityUser	Optional instance of the current ProjectActivityUser object
bp	The bp environment
bpEventHandle	The class that holds information about the event that generated postback
bpEventHandle.EventType	The type of event
	bp.EventType.User – A custom button was hit
	bp.EventType.Complete – A process complete button was hit
	bp.EventType.Cancel – The cancel button was hit
	bp.EventType.Save – The Save button was hit
	bp.EventType.SaveAndClose – The Save and Close button was hit
	bp.EventType.Print – The Print button was hit
	bp.EventType.CancelClose – The Cancel process button was hit
bpEventHandle.EventName	The name of the control that initiated the postback
bpEventHandle.EventControl	The actual FormControl of the control that initiated the postback

LOCAL VARIABLE	DESCRIPTION
bpEventHandle.EventControl.ArrayNum	The array row number of event, or 0 if event was not caused inside array.

The Form life cycle has the following event callbacks:

```csharp
<%@ Control Language="C#" AutoEventWireup="true"
    Inherits="BPLogix.WorkflowDirector.SDK.bpFormASCX" %>
<script runat="server">
// These methods are optionally overridden to allow custom
script
// to be inserted into the Form life-cycle.

// Called 1 time per form instance to initialize form fields
protected override void BP_FormInitialize()
{
}

// Called first time in ViewState that form is displayed
protected override void BP_ViewStateInit()
{
}

// Called for every event control and complete button
protected override void BP_Event(bp.EventType pEventType,
                                  string pEventName)
{
}

// Called prior to processing rules
protected override void BP_Rules()
{
}

// Called after processing rules
protected override void BP_Rules_Post()
{
}

// Called prior to completing before internal validation
protected override void BP_Validation()
{
}

// Called prior to completing after internal validation
protected override void BP_Validation_Post()
{
```

```
}

// Called prior to saving form data and closing form
protected override void BP_Completed()
{
}

// Called just prior to displaying a form
protected override void BP_Display()
{
}
</script>
```

These methods are optionally overridden to allow custom script to be inserted into the Form life-cycle:

EVENT	DESCRIPTION
`BP_ FormInitialize ()`	Called 1 time per form instance to initialize form fields
`BP_ViewStateInit()`	Called first time in ViewState that form is displayed
`BP_ Event (bp.EventType pEventType, string pEventName)` See sample above for example.	Called for every event control and complete button {table below}

pEventType:	The type of control that caused the event
`bp.EventType.User`	A user event from a form control
`bp.EventType.Complete`	One of the complete buttons on a form
`bp.EventType.Save`	Save form data
`bp.EventType.SaveAndClose`	Save form data and close form
`bp.EventType.Print`	Print form
`bp.EventType.CancelClose`	Cancel process or Form submission
`bpEventName`	The ID of the form

EVENT	DESCRIPTION	
	pEventType:	**The type of control that caused the event**
		control that caused the event
`BP_Rules()`	Called prior to processing rules	
`BP_Rules_Post()`	Called after processing rules	
`BP_Validation()`	Called prior to completing before internal validation	
`BP_ Validation_ Post ()`	Called prior to completing after internal validation	
`BP_Completed()`	Called prior to saving form data and closing form	
`BP_Display()`	Called just prior to displaying a form	

Process Scripts

This section documents how to write custom process scripts for Process Director. A process script is called from a Script activity in a Process Timeline definition. The Script task type specifies a Script Function Name. This function name can a custom script function. Your custom functions can be located on a specific custom script file (in the Process Director database). Each step in the process definition points to a script file.

To develop process Scripts inside Visual Studio, use the fully functional Visual Studio project installed with the product named bpVS.zip.

Using a Specific Process Script File

To write a custom script for a process, create a script file with ".ASCX" as the file extension on your local hard drive. Add your custom script code and then upload the file to the Process Director database using the Create New menu item in the Content List (select Document/File in the dropdown). Browse to the location of your ASCX file and upload the file to Process Director. This file will be displayed in the Content List on the server. To make changes to this script file, you must check out the file first, update it, and then upload the new version. For information about modifying files and documents on Process Director, refer to the Implementers

Reference Guide. The script for a process is configured in the Properties page of the process definition. The script must exist in the Process Director database and it will be called when a Script task is run in the Process Timeline. For more information on configuring process definition properties refer to the Process Timeline topics in the Implementers Reference Guide.

Writing a Process Script Function

The optional parameters specified in the Script task are used to call your Script function in the process custom script file. For example if you specify "SOME_PARM" as the Script Parameters specified in the Script task, the parameter to your C# function (pParm in the example below) will be set to "SOME_PARM". If you need to pass multiple values to the process script, you'll need to encode the values by separating them by commas or other techniques (e.g. MY_PARM,1,2). The entire string will be passed to the process script function as a single parameter. You'll need to parse the string into its components using C# string manipulation functions. Your function in the script file would look as follows:

```
<%@ Control Language="C#" AutoEventWireup="false"
    Inherits="BPLogix.WorkflowDirector.SDK.bpFormASCX" %>
<script runat="server">
    public override void Process_Script(string pParm)
    {
        bp.log0("Called Process script:" + pParm);
    }
</script>
```

This function is immediately available to all processes as soon as the file is saved. You may also use System Variables in the Script Step parameter field to extend the capabilities of the process script.

This method will be called with the following environment:

LOCAL VARIABLE	DESCRIPTION
CurrentWorkflow	The current Workflow object (deprecated)
CurrentWorkflowStep	The current WorkflowStep object (deprecated)
CurrentForm	Optional Form instance object of the process form
CurrentPartition	Instance of the current Partition object

LOCAL VARIABLE	DESCRIPTION
CurrentProject	Optional instance of the current Project object
CurrentProjectActivity	Optional instance of the current ProjectActivity object
bp	The bp environment

Debugging Process Scripts

To test your custom script, run the process that contains the Script task that calls your function. When that step in the process is run, any error will cause the step to remain running and never complete. View this step using the Timeline/Administration tab of the Timeline instance.

If any errors are encountered running the script, the step will stop with and have the status set to Error. The status field of the step will show the exact error (e.g. a compile error for the script). Correct any errors in your custom script and then right click the Timeline Activity in the Timeline/Administration tab of the Timeline instance, and select Restart Activity. When your script runs successfully the activity will complete and transition to the next step in the process.

You can also debug a process script using the logging functions, like bp.log0 and bp.log1.

Process Script Handlers

This section documents how to write custom script handlers for Process Director which are called prior to each process step starting, when an error is encountered, and when the process step ends. Process Timelines have similar script events that can be incorporated into the process script. You can configure a single script file in the process definition by using the Advanced Options tab of the Settings.

Scripted behavior can be added to the beginning or end of any Process Timeline Activity. The script events `Timeline_StartActivity` and `Timeline_StopActivity` events are used for Timelines. The `Timeline_StartActivity` event is called prior to the start of each Timeline Activity. Similarly, the `Timeline_StopActivity` event IS called when each Timeline Activity ends.

Using a Specific Process Script File

To write a custom script for a process, create a script file with ".ASCX" as the file extension on your local hard drive. Add your custom script code and then upload

the file to the Process Director database using the Create New menu item in the Content List (select Document/File in the dropdown). Browse to the location of your ASCX file and upload the file to Process Director. This file will be displayed in the Content List on the server. To make changes to this script file, you must check out the file first, update it, and then upload the new version. For information about modifying files and documents on Process Director, refer to the Implementers Reference Guide. This script for a process is configured in the Properties page of the process definition. The script must exist in the Process Director database and it will be called when a Script task is run in the process. For more information on configuring process definition properties refer to Workflow and Timeline chapters in the Implementers Reference Guide.

Writing the Script Handler

Your function in the script file would look as follows:

```
<%@ Control Language="C#" AutoEventWireup="false"
    Inherits="BPLogix.WorkflowDirector.SDK.bpFormASCX" %>
<%@ Import Namespace="System.Collections.Generic" %>
<script runat="server">
    // these functions are used for Timeline activities
    public override void Timeline_Script(string pParm)
    {
        bp.log0 ("Timeline_Script: " + pParm);
    }
    public override void Timeline_PreActivity(List<Pro-
cessUser> pUsers)
    {
        bp.log0 ("Timeline_PreActivity");
    }
    public override void Timeline_StartActivity()
    {
        bp.log0 ("Timeline_StartActivity");
    }
    public override void Timeline_StopActivity()
    {
        bp.log0 ("Timeline_StopActivity");
    }
    public override void Timeline_Check()
    {
        bp.log0 ("Timeline_Check");
    }
</script>
```

These functions are immediately available to all processes as soon as the file is uploaded to the Content List.

`Timeline_StartActivity` will be called at the beginning of every step in the process, even if the step has no users. It is called before `Timeline_PreActivity`.

`Timeline_PreActivity` will be called prior to every step in this process starting. The list of users which will be assigned to the `CurrentProjectActivity` is passed in the `pUsers` parameter. You can add to, remove, or replace users in this list. `Timeline_PreActivity` is only called for user steps.

`Timeline_StopActivity` is called prior to a step's completion or termination.

`Timeline_Check()` enables you to check for advancing and due date/timer processing, and can be useful to implement custom timer processing. These are optional overrides that can be defined in the script associated with the Process Timeline.

When a custom timeline script alters the state of a process, it will be invoked immediately after the script runs, so that changes made to the process state are immediately reflected to the user.

PROCESSUSER PROPERTIES	DESCRIPTION
UID	The UID of the user who will be assigned to this step
SUID	Optional STEP USER ID that is starting this user, could be a group step user record
AdminUID	Optional, used to set the UID of the person adding the user to this step
AdminComment	Optional, used to set the comment from the person adding the user to this step

This method will be called with the following environment:

LOCAL VARIABLE	DESCRIPTION
`CurrentWorkflow`	The current Workflow object (Deprecated)
`CurrentWorkflowStep`	The current WorkflowStep object (Deprecated)
`CurrentForm`	Optional Form instance object of the process form
`CurrentPartition`	Instance of the current Partition object

LOCAL VARIABLE	DESCRIPTION
`CurrentProject`	Optional instance of the current Project object
`CurrentProjectActivity`	Optional instance of the current ProjectActivity object
`bp`	The bp environment

Knowledge View Scripts

This section provides a reference for writing custom Knowledge View scripts for Process Director. The Knowledge View supports a custom script that can be called before the results are displayed to the user. The custom scripts are stored in the Process Director database in the Content List. The custom script is used to inspect, modify, calculate, or remove results from the Knowledge View.

To develop Knowledge View Scripts inside Visual Studio, use the fully functional Visual Studio project installed with the product named bpVS.zip. Refer to the sample file kview_script.ascx.

Writing a Knowledge View Script Function

A Knowledge View can have a custom script that can alter the results of the displayed information.

When the Knowledge View runs, it will call this method in the script for EVERY row in the result:

```
public override bool KV_Display(List<NameValueEx> pColumns,
                                ContentObject pObject,
                                out bool pRemoveRow)
```

If you set the pRemoveRow to true, then the row being processed will be excluded from the report.

The example below will add the HTML "bold" tag around the value for the Amount columns.

```
<%@ Control Language="C#" AutoEventWireup="false"
    Inherits="BPLogix.WorkflowDirector.SDK.bpScript" %>
<%@ Import Namespace="System.Collections.Generic" %>
<%@ Import Namespace="BPLogix.WorkflowDirector.SDK" %>
<script runat="server">
    public override bool KV_Display(List<NameValueEx>
```

```
pColumns,
                                ContentObject pObject,
                                out bool pRemoveRow)
    {
        pRemoveRow = false;// Do not remove the row being pro-
cessed
        foreach (var entry in pColumns)
        {
            // Bold the Amount column
            if (entry.Name == "Amount")
            {
                entry.Value = "<b>" + entry.Value + "</b>";
            }
        }
        return true;// Return true if a value has changed
    }
</script>
```

Notice that Knowledge View scripts inherit from BPLogix.WorkflowDirector.SDK.bpScript.

This method will be called with the following environment:

LOCAL VARIABLE	DESCRIPTION
CurrentUser	Optional instance of the current User object
CurrentPartition	Instance of the current Partition object
bp	The bp environment

NameValueEx List Object

Note that the Knowledge View script uses the `NameValueEx` list object to store the values in each of the Knowledge View columns. The Value attribute of the `NameValueEx` list is always a string, but, of course, the value may be derived from a non-string object. Let's say that a value we wish to find comes from a checkbox. In that case, we would see the Value of the checkbox represented by the string values "True" or "False". If the Value contains a string representation of a number, on which you'd like to do some math, you can use the built-in Process Director conversion methods to convert the Value string to an appropriate numeric value.

In addition to the Value attribute, however, the `NameValueEx` list class also has a `ValueEx` attribute that contains the actual .NET DataItem object for the field, or a decminal number.

The `NameValueEx` object constructor has four overloads:

```
//Null values
public NameValueEx()

//String Name and Value
public NameValueEx(string pName, string pValue)

//String Name and Value, and Decimal Number
public NameValueEx(string pName, string pValue, decimal pNum-
ber)

//String name and Value, and List DataItem
public NameValueEx(string pName, string pValue, DataItem
pDataItem)
```

Custom ASPX Pages

You can extend the BP Logix web application by providing your own custom .ASPX pages (typically placed into the /custom folder). These pages can call BP Logix SDK APIs, or perform any other logic. This page can be used, for example, to perform scheduled logic using the Windows Scheduler. Or you can write custom pages that are called from external applications or portals.

See the sample in the /custom/samples/SamplePage.aspx.sample file. Notice that the page is derived from the BPLogix.WorkflowDirector.SDK.bpCustomPage class. This class enables you to call any BP Logix SDK API from the new page. Your code will typically be placed into the Page_Load event.

To create custom ASPX pages inside Visual Studio, use the fully functional Visual Studio project installed with the product named bpVS.zip. Refer to the sample file SamplePage.aspx.

A more detailed explanation of creating Custom ASPX Form pages is provided in the Creating ASP.NET Forms topic.
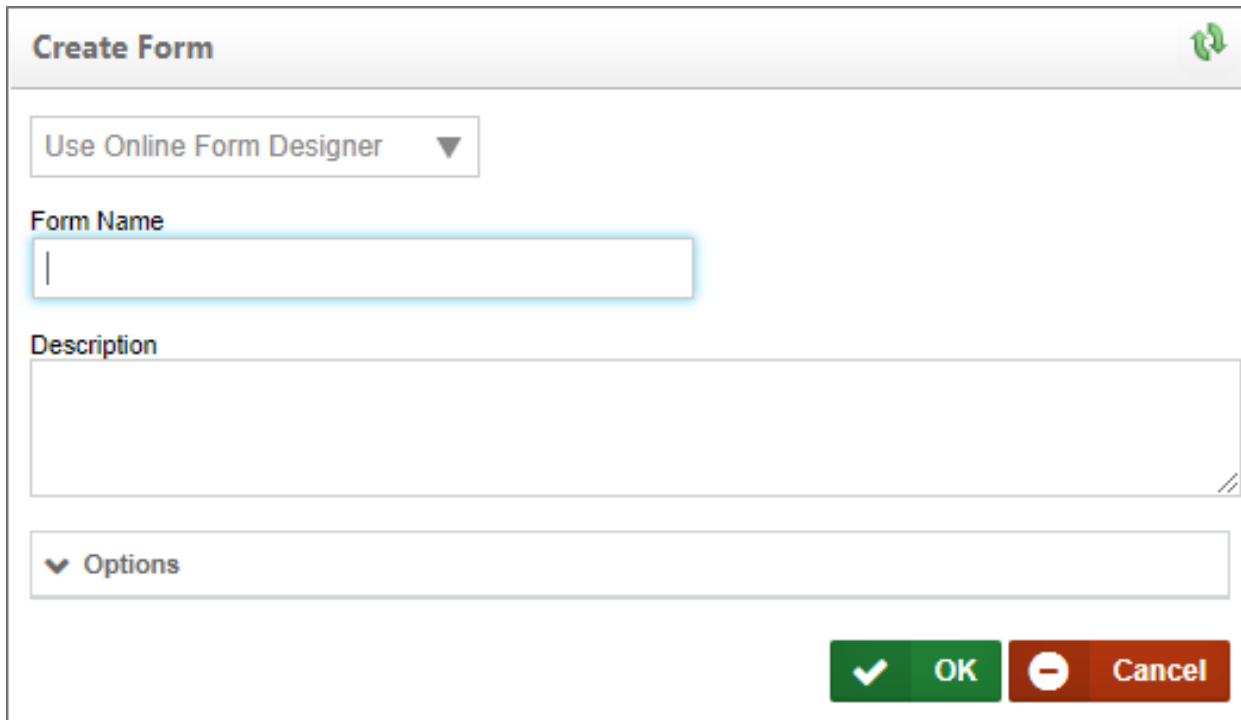
Creating ASP.NET Forms

This section will describe how to create and manage a Form, how to set the different variables associated with the Form form fields, and how to implement your custom scripting. You'll notice some differences from previous versions, such as no more Form page refresh. Forms now utilize AJAX which eliminates the page refresh when using events. Please keep in mind; this is for Form development using ASP.Net. We won't be using the Form builder for this section.

For more information about the Form and its properties please refer to the [Imple-menters Reference Guide](#).

Adding a New Form Definition

Process Director can create new Forms as ASCX controls for you. Simply select Form Definition from the Create New dropdown menu to open the Create Form screen.

You have three options to create the ASCX control Form from the Create Form screen. You'll find the options in the dropdown at the top of the Create Form screen.
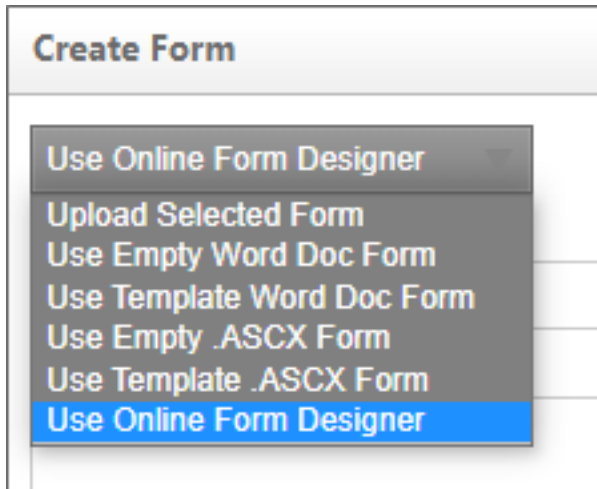
The three relevant options from which to choose are *Upload Selected Form*, *Use Empty .ASCX Form*, and *Use Template .ASCX Form*, all of which are discussed below.

Option 1: Upload Selected Form

To create a new Form definition, you'll have to create an .ascx page using your development tool. There is one line of code that must be included before you upload your new page. Please copy and paste the following in the first line of your page:

```
<%@ Control Language="C#" AutoEventWireup="false" Inher-
its="BPLogix.WorkflowDirector.SDK.bpFormASCX"%>
```

Once you've created your Form, you can upload it by selecting Form Definition from the Create New menu. You'll be presented with a page to browse and select your form file. You'll also have the option to provide a Name and Description for the Form definition.

Option 2: Use Empty .ASCX Form

If you use this option, Process Director will create an ASCX control that contains only basic code required for the control. The basic code consists only of the document declaration at the top of the page, and a <script> tag that contains the stubs for the common Process Director scripting methods.

Once you've created the control, you can Check Out the Form from the Edit tab of the Form definition, and begin working on it in your development environment.
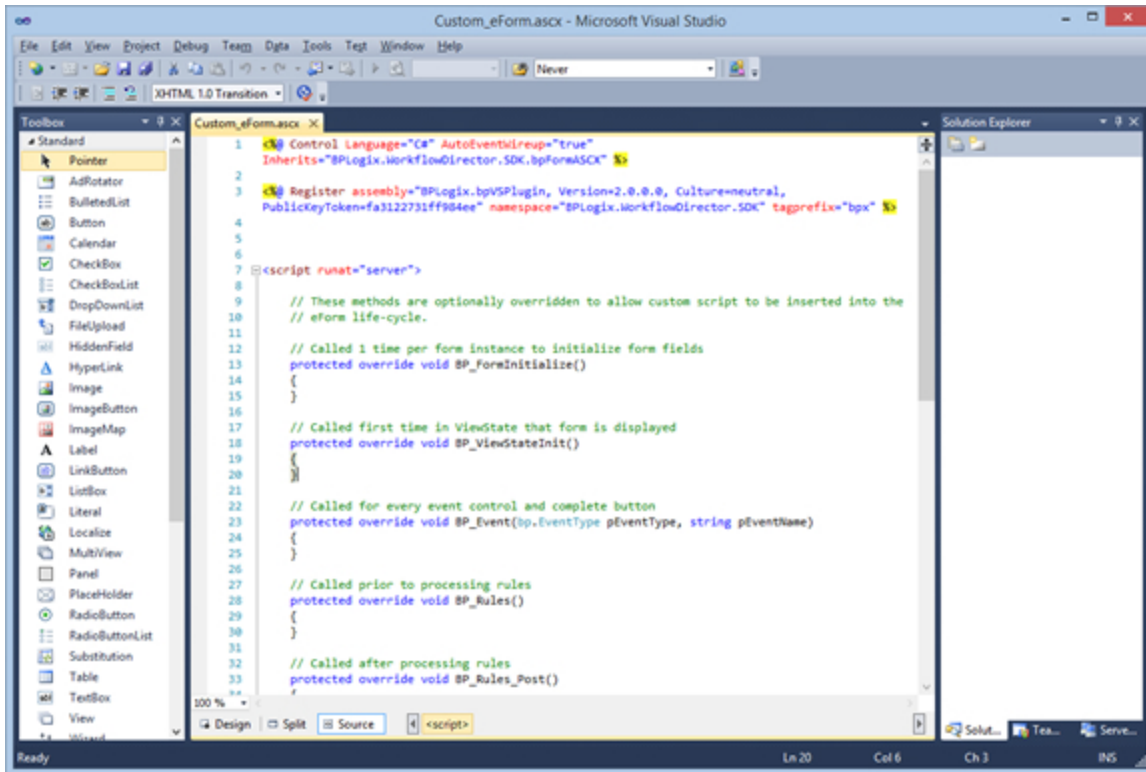
Option 3: Use Template .ASCX Form

If you use this option, Process Director will create an ASCX control that contains a sample Form with some controls, formatting styles, and other HTML content, in addition to the page declaration and script stubs.

Again, you can edit this form in your development environment after checking out the form from the Edit tab of the Form definition.
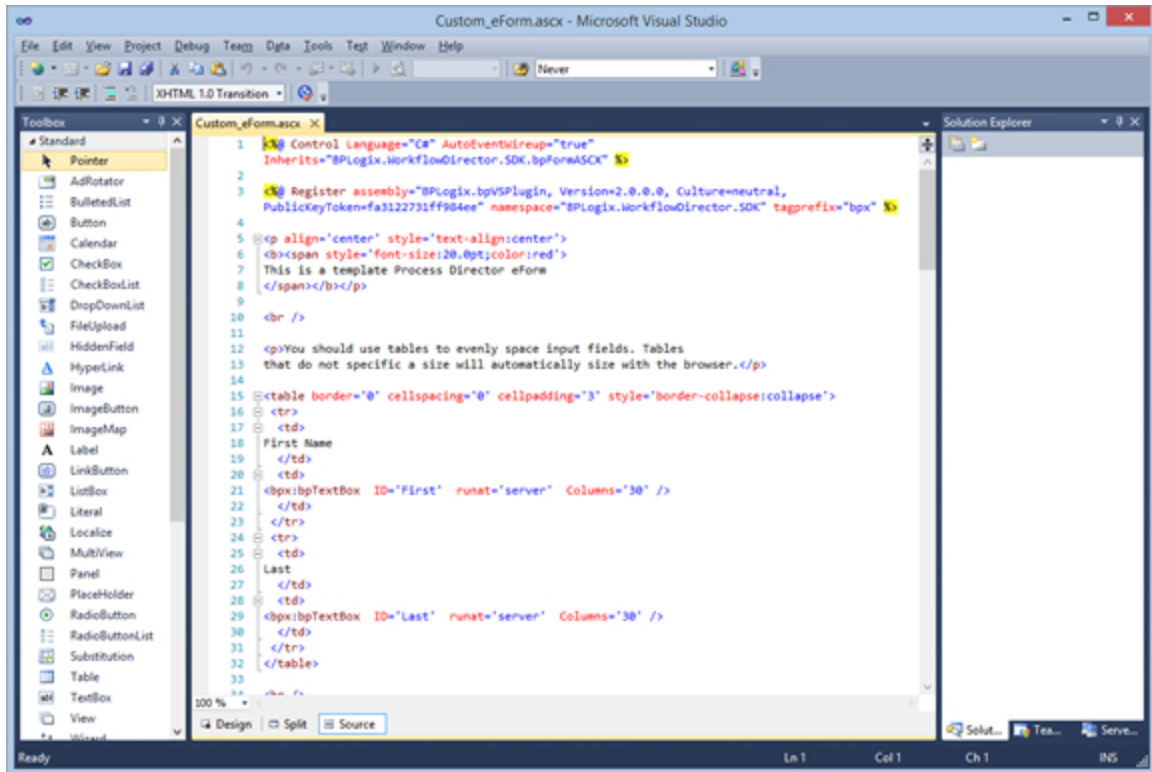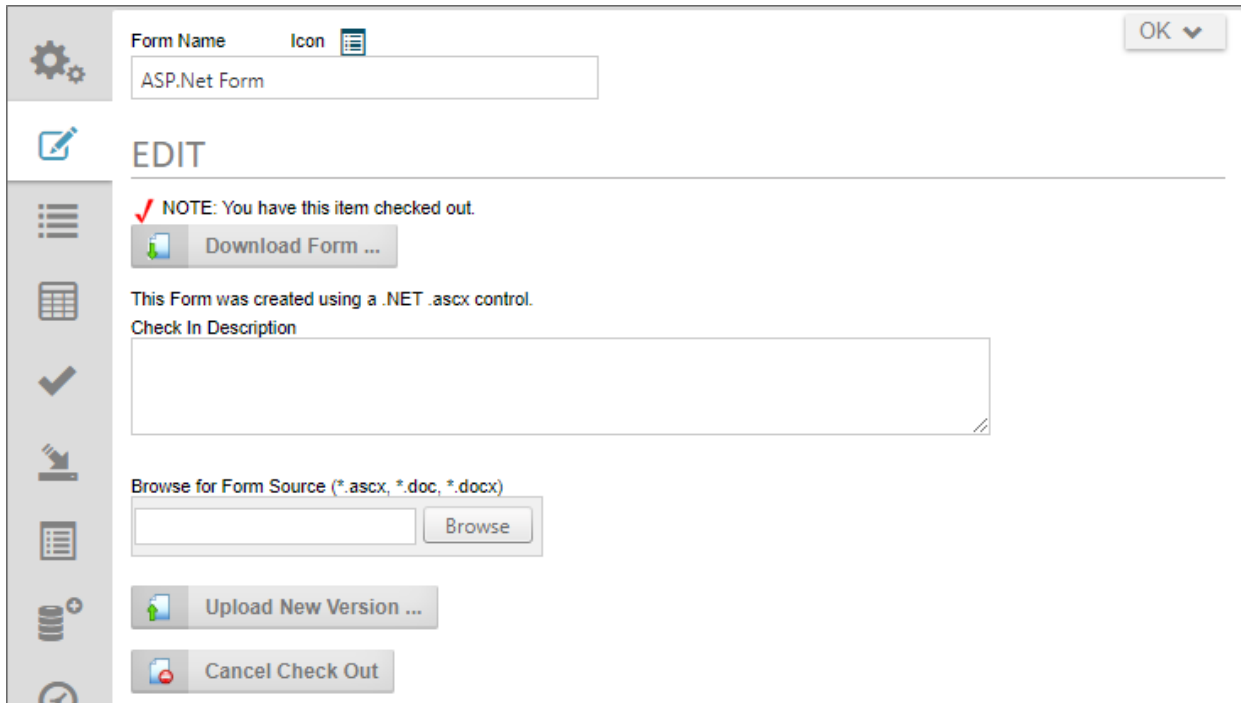
Editing an ASP.NET Form

ASP.NET Forms can, once created, be edited by navigating to the Edit tab of the Form definition.
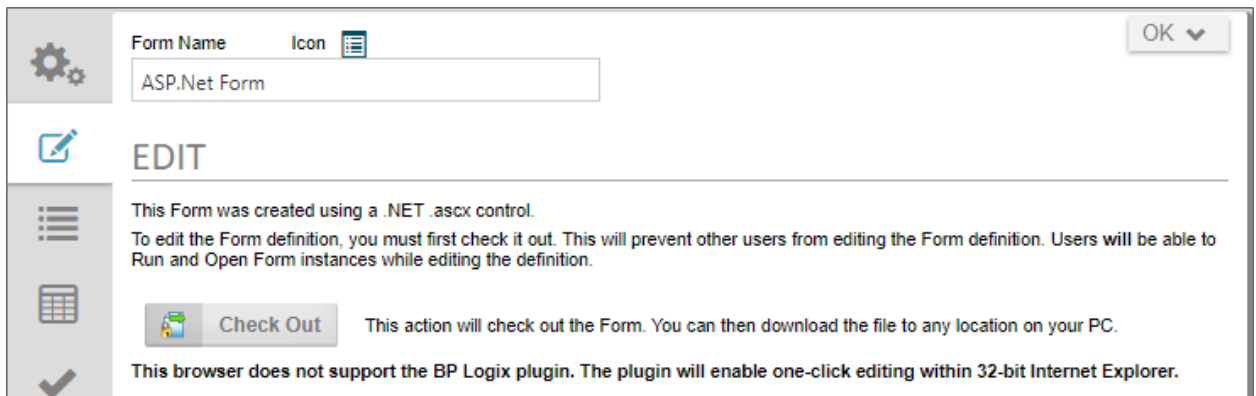
Check out the ASCX form by clicking the Check Out button. Doing so will check out the form for editing and give you access to the editing features of the form definition.

To edit the form in Visual Studio, you can download it by clicking the Download Form button to download the form to your local computer. You can also stop the editing process at any time, and revert to the currently saved version of the form by clicking the Cancel Check Out button.

Once you're done editing the form, you can provide the appropriate text for the Check In Description, then click the Browse button to find the editied version of the form on your local computer and select it for upload. Once you've done so, click the Upload New Version button to upload and check in the edited form, which will replace the existing form with the newly edited version.

Developing a Form in the .NET environment

There are two ways to develop Forms within Process Director. The first uses the Online Form Designer that's documented in the Implementers Reference. The other approach is to develop native ASP.NET forms, which is described here.

To develop Forms inside Visual Studio, use the fully functional Visual Studio project installed with the product named bpVS.zip. Refer to the sample files eform_*.aspx.

This section will provide you the basics of developing your Form in Visual Studio 2008. Developing in this environment requires the knowledge to program in ASP.NET. You'll be able to use ASP.NET controls as well as extended controls created by BP Logix.

When creating a Form for Process Director, you are actually creating a custom control (.ascx). You create your Form just as if you were creating in the ..ASPX page. Below you'll see the basic structure of the Form.

```
<%@ Control Language="C#" AutoEventWireup="true" Inher-
its="BPLogix.WorkflowDirector.SDK.bpFormASCX"%>
<script runat="server">
// Add any events here that will be called as part of the Form
// life-cycle
// See the Custom Scripting / Form Scripts section for more
information
</script>
```

Include Files

This enables common functions to be called from other script files. To include a file in your Form use the following syntax:

```
<!--#include file="~/Custom/MyScripts/script.ascx"-->
```

The .ascx must exist on the server file system and isn't controlled by Process Director (via the Content List). A good location for an include file is the /custom/ folder in the Process Director web site installation directory.

Using a .DLL file with Your Scripts

This is an approach to call code from custom scripts. Developers can make a "normal" .NET .dll and place it into the \Program Files\BP Logix\Process Director\website\bin folder on the server. Then they can reference the classes in that .dll through any script code in the Process Director application. You can use the GAC, but it is easier to use the bin folder (assuming the code wouldn't be used "outside" of the Process Director environment). DLLs in this folder are accessible only to the Process Director application.

In your module placing the public classes into a namespace, such as:

```
namespace companyname.custom
{
    public class MyClass()
    {
        // … your methods, properties, etc
        public static void MyFunc()
        {
        }
```

```
    }
}
```

Then inside a form script call this function using:

```
companyname.custom.MyClass.MyFunc();
```

Form Controls

This section outlines the various Process Director controls that can be placed onto Forms. These controls are used to create and enhance Forms in addition to "normal" ASP.NET controls (such as TextBox, DropDownList, etc). The properties can be set in the actual `<bpx>` control tag, or can be set via normal C# properties.

AddRow

This Form control will create a button a user can click to add row(s) to an array.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
ArrayName	The name of the array this button is attached to.	
At	The location to add the new row (s).	0 (end of array).
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.	
ImageURL	Sets an optional image for the button.	
OnClientClick	Used to execute client-side JavaScript or call client JavaScript functions. To prevent the button from causing a Post-back, place a return false; at the end of the JavaScript string.	

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Rows	The number of Rows to add.	1
Text	Sets the optional button text.	

Example

```
<bpx:AddRow runat="server" ArrayName="MyArray"/>
```

Array

This Form control places a repeating template section on a Form. To default the array to a number of rows simply go to the Form properties page and click on the edit link next to the array control that is in the list of controls. Select Value → Number from the Default Value dropdown and enter the number to default the number of rows to.

Properties

None

Example

```
<table>
    <tbody>
        <tr>
            <th>City</th>
            <th>State</th>
        </tr>
        <bpx:Array ID="ArrayTest" runat="server">
        <ItemTemplate>
        <tr>
            <td><bpx:bpTextBox ID="Text1" runat="server"
/></td>
            <td><bpx:bpTextBox ID="Text2" runat="server"
/></td>
        </tr>
        </ItemTemplate>
        </bpx:Array>
    </tbody>
</table>
```

ArrayRemoveRow

This Form control will create a button a user can click to remove a specific row from an array. You should place this control directly in an array, so that the button

is displayed on each row.

Properties

PROPERTY NAME	DESCRIPTION
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.
ImageURL	Sets an optional image for the button.
OnClientClick	Used to execute client- side JavaScript or call client JavaScript functions. To prevent the button from causing a Post-back, place a return false; at the end of the JavaScript string.
Text	Sets the optional button text.

Example

```
<bpx:ArrayRemoveRow runat="server"/>
```

ArrayMoveUp

This Form control will create a button a user can click to move a Row up in the array.

Properties

PROPERTY NAME	DESCRIPTION
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.
ImageURL	Sets an optional image for the button.
OnClientClick	Used to execute client- side JavaScript or call client JavaScript functions. To prevent the button from causing a Post-back, place a return false; at the end of the JavaScript string.
Text	Sets the optional button text.

Example

```
<bpx:ArrayMoveUp runat="server"/>
```

ArrayMoveDown

This Form control will create a button a user can click to move a Row down in the array.

Properties

PROPERTY NAME	DESCRIPTION
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or con- firm the action which the button will take.
ImageURL	Sets an optional image for the button.
OnClientClick	Used to execute client- side JavaScript or call client JavaScript functions. To prevent the button from causing a Post-back, place a return false; at the end of the JavaScript string.
Text	Sets the optional button text.

Example

```
<bpx:ArrayMoveDown runat="server"/>
```

Attach

This Form control will display a button to allow the user to attach files to the form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
AttachType	Form Workflow Timeline Process	Attach object (s) directly to Form, Attach object (s) to the cur-	Process

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
		rent Workflow instance as a Workflow reference. Attach object(s) to the current timeline instance as a Timeline reference. Attach object(s) to the current Workflow or timeline instance.	
ClipboardImageName		The optional name to use for the attached images. This parameter can use SysVars to make the name dynamic. If the items on the clipboard are files, the actual file name will be used as the new attach-	

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
		ment name.	
ConfirmText		Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.	
GroupName		Optional name of the group to place the attachment (s) into.	
ImageURL		Sets an optional image for the button.	
ObjectType	Document Clipboard	Allow user to upload document Allow the user to use an item from the clipboard	Document
OnClientClick		Used to execute client-	

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
		side JavaScript or call client JavaScript functions. To prevent the button from causing a Postback, place a return false; at the end of the JavaScript string.	
SingleFileUpload		When set to "true", restricts the upload to a single file.	false
Text		Sets the optional button text.	

Example

```
<bpx:Attach ID="ControlName" runat="server"
GroupName="Group"/>
```

AttachKview

This control enables the user to attach an object to a Form instance by browsing for the object using a Knowledge View.

Properties

PROPERTY NAME	DESCRIPTION
AttachToParent=[1|0]	If set to 1, this object will be attached to the

PROPERTY NAME	DESCRIPTION
	parent of this Form instance
AttachType	Form – attach object(s) directly to Form
	Workflow - attach object(s) to the current Workflow instance as a Workflow reference
	Process Timeline - attach object(s) to the current Process Timeline instance as an Process Timeline reference
	Process - attach object(s) to the current Workflow or Process Timeline instance
CopyObject=[1|0]	If set to 1, this object will be copied to the new location (leaving the old object alone)
GroupName	Only display objects from the specified group
MoveObject=[1|0]	If set to 1, this object will be moved to a new location (removing the object in its old location)
QS	A querystring to send data to the Knowledge View
Text	Text displayed on the Form button

Example

```
{AttachKVIew:ControlName, AttachType=Form, GroupName=Group,
    Text=Text to Display, MoveObject=0}
```

AttributePicker

This Form control will create a picker control to allow users to pick meta data attributes.

Properties

PROPERTY NAME	DESCRIPTION
StartingCategory	Sets the category from which to show the initial attributes.
Text	Sets the optional button text.

Example

```
<bpx:AttributePicker runat="server"/>
```

bpButton

This Form control is used to place a button on a Form. The button will typically be used when hooking up to a Custom Task, or when writing custom C#.

Properties

PROPERTY NAME	DESCRIPTION
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or con-firm the action which the button will take.
OnClientClick	Used to execute client-side JavaScript or call client JavaScript functions. To prevent the button from causing a Post-back, place a return false; at the end of the JavaScript string.
StartingCategory	Sets an optional image for the button.
Text	Sets the optional button text.

Example

This example will cause a Post-back to the server to process the event.

```
<bpx:bpButton ID="ControlName" Text="Text to Display" run-
at="server" />
```

This example will execute client JavaScript, and prevent a Post-back.

```
<bpx:bpButton ID="ControlName" Text="Text to Display"
    OnClientClick="MyFunc();return false;" runat="server" />
```

bpCheckBox

This Form control places a two-state (checked - true/unchecked - false) checkbox on the Form. Useful for yes/no data and enabling/disabling sections on a Form.

Properties

PROPERTY NAME	DESCRIPTION
CssClass	To set the CSS class name for this control.
Text	(optional) Accompanying label text for the check box

Example

```
<bpx:bpCheckBox ID="ControlName" runat="server" Text="Text to
Display" />
```

bpImage

This Form control places a configurable image control on the Form. The ImageURL property is required, but all other properties are optional.

Properties

PROPERTY NAME	DESCRIPTION
ImageURL	The fully qualified URL of the source image's location.
Height	Height of the image in pixels/percent.
Width	Width of the image in pixels/percent.
URL	The fully qualified URL of an asset to display when the image is clicked, i.e., to make the image a hyperlink.
Target	The hyperlink target type, e.g. "_blank" for the image hyperlink.
Style	Any CSS styles to apply to the image
CssClass	To set the CSS class name for this control.
Alt	The alt text for the image.

Example

```
<bpx:bpImage ImageURL="imgURL" Height="XX" Width="XXX"
URL="HyperlinkTargetURL" Target="HyperlinkTarget"
    Style="CSSStyles" CssClass="ClassName" Alt="Alt Image
Text" />
```

bpLabel

This Form control will simply display Text. This can be used, for instance, when you'd like to conditionally show text. You can associate rules with this control to affect the visibility. You'll use C# to get/set the actual Text contents.

Properties

PROPERTY NAME	DESCRIPTION
CssClass	To set the CSS class name for this control.

Example

```
<bpx:bpLabel runat="server" ID="ControlName"/>
```

bpString

This Form control will display Text from a localized resource file. You can create custom localized strings in the strings.resx files to easily display strings in different languages on forms. You can associate rules with this control to affect the visibility.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
CssClass	To set the CSS class name for this control.	
DefaultString	String display while editing form definition in Design mode using VS plugin	
ResourceID	Name of resource in strings.resx or resource.resx	
ResourceType	Does string exist in strings.resx (Custom) or resource.resx (Internal)	Custom

Example

```
<bpx:bpString runat="server" DefaultString="StringValue"
    ResourceID="ResIDString" />
```

bpTextBox

This Form control puts a space for a user to enter text on a Form.

Properties

PROPERTY NAME	DESCRIPTION
Columns	To set the number of columns the control will use.
CssClass	To set the CSS class name for this control.
Rows	To set the number of rows the control will use.
TextMode	SingleLine, Multiline, or Password

Example

This example will cause a Post-back to the server

```
<bpx:bpTextBox ID="ControlName" runat="server" Columns="NN"/>
```

ButtonArea

This Form control is used to control where the complete buttons for the Form are placed. Complete buttons are buttons such as OK, Cancel, Approve, Reject etc. The actual buttons that are placed in this area are dependent on the current Timeline Activity, if any. If this control isn't present on a Form, then the buttons are added to the bottom of the form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
CancelConfirmText		Pops up a confirmation box when a user clicks the Cancel button with the specified text, allowing a user to return to the form or cancelling out of the form.	

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
CancelImageURL		Optional path to the image used for the Cancel button on a Form	
CancelShow	True False	Enables the form to show or hide the Cancel button.	True
CancelText		Sets the text for the Cancel button	Cancel
CompleteConfirmText		Pops up a confirmation box when a user clicks the Complete button with the specified text, allowing a user to cancel the Complete or continue to submit the form.	
CompleteShow	True False	Enables the form to show or hide the Complete button.	True

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
OKConfirmText		Pops up a confirmation box when a user clicks the OK button with the specified text, allowing a user to cancel the OK or continue submitting the form.	
OKImageURL		Optional path to the image used for the OK button on a Form	
OKShow	True False	Enables the form to show or hide the OK button.	True
OKText		Sets the text for the OK button	OK

Example

This example controls the location of the complete buttons, and changes the default text of the OK and Cancel buttons.

```
<bpx:ButtonArea runat="server" OKText="OK Text"
    CancelText="Cancel Text"/>
```

Calculate

This Form control calculates an expression and places the result as text on a Form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
FormatString		(optional) The format in which to display the result of the Formula (Defaults to " {0:0.00}") - See [Microsoft's](#) [documentation](#) on string formatting
Formula		Expression to calculate a numerical value (can accept System Variables)

Example

```
<bpx:Calculate ID="ControlName" runat="server"
    Formula="{#FORM:Field1} * {#FORM:Field2}"
    FormatString="{0:0.0}" />
```

Cancel

This Form control will place a button on a Form which will cancel or delete the current Form or Process Timeline.

Properties

PROPERTY NAME	DESCRIPTION
CancelProject	This enables you to cancel the associated timeline.
CancelWorkflow	This enables you to cancel the associated Workflow.
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.
DeleteForm	This enables you to delete the form.

PROPERTY NAME	DESCRIPTION
DeleteProject	This enables you to delete the timeline.
DeleteWorkflow	This enables you to delete the Workflow.
ImageURL	Sets an optional image for the button.
Text	Sets the optional button text.

Examples

To cancel associated Process Timeline:

```
<bpx:Cancel runat="server" CancelProject ="true"
    Text="Text to Display">
</bpx:Cancel>
```

To just delete this form:

```
<bpx:Cancel runat="server" DeleteForm="true"
    Text="Text to Display">
</bpx:Cancel>
```

To delete the Process Timeline and Form:

```
<bpx:Cancel runat="server" DeleteProject ="true"
    DeleteForm="true" Text="Text to Display">
</bpx:Cancel>
```

CategoryPicker

This Form control will create a picker control to allow users to pick meta data categories.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Multiple	Enables you to select multiple categories from the picker.	"false"
StartingCategory	Sets the initial category to display.	
Text	Sets the optional button text.	

Example

```
<bpx:CategoryPicker runat="server"/>
```

CommentLog

This Form control enables a user to place a Comment Log on a Form.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Columns	(Optional) The number of columns to display while prompting for a comment	70
ControlName	(Optional) The name of the comment log section. Use this property if you have multiple comment logs on a Form	
Rows	(Optional) The number of rows to display while prompting for a comment	4
Text	(Optional) The text for the button used to add a comment	Add Comment
Width	(Optional) Width of the displayed comments	100%

Example

```
<bpx:CommentLog runat="server" ControlName="ControlName"
Width="50%"/>
```

ContentPicker

This Form control enables a user to choose an object in the content list.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
DocExtension		(Optional) Limits the user to choose only	

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
		documents of the specified extension (Type of "Document" only)	
StartingFolder		(Optional) The path to a folder, limiting a user to choose only objects in that folder and its subfolders	
Type		(Optional) The type of object (Folder, ContentObject, Script, etc.) to pick.	Folder

Example

```
<bpx:ContentPicker ID="ControName" runat="server"
    Type="Document" DocExtension="pdf" />
```

ControlPicker

This Form control will display a dropdown of all controls on this page.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
ControlType	Input, Textarea, Date, Button, Dropdown, Password, Array, Section, Radio,	Limits the type of control to show in the dropdown.

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
	CheckBox, Custom, CustomTaskConfigSection, CustomTaskRunSection, UserPicker, GroupPicker, Attach, ShowAttach, Label.	
DropdownPrompt		Optional text to show on the dropdown if no user is selected
Style	To set the style (using any CSS style).	

Example

```
<bpx:ControlPicker ID="ControlName"
    DropdownPrompt="Prompt Text"
    runat="server" />
```

DatePicker

This Form control is a date picker control.

Properties

PROPERTY NAME	DESCRIPTION
BlockControl	If set to true, will surround the control within an HTML block element.
Style	To set the style (using any CSS style).

Example

```
<bpx:DatePicker ID="ControlName" runat="server"/>
```

DateTimePicker

This Form control is a date/time combination picker control.

Properties

PROPERTY NAME	DESCRIPTION
EndTime	(Optional) Sets the maximum time (of day) for the pre-selected picker values. Must exceed the StartTime value.
Interval	(Optional) The amount of time (in minutes) between pre-selected picker values.
StartTime	(Optional) Sets the beginning time (of day) for the pre-selected values available for the picker.

Example

```
<bpx:DateTimePicker ID="ControlName" runat="server" />
```

DateDiff

This Form control is used to calculate the difference between 2 dates.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Date1	The first date or date / time	
Date2	The second date or date / time	
Type	(Optional) The type of the difference Years Months Days BusinessDays Hours BusinessHours Minutes Seconds	Days

Example

```
<bpx:DateDiff ID="ControlName" Date1="1/1/2000" Date2="
{form:my_date2}"
    runat="server"/>
```

DBConnectorPicker

This Form control will display a Database Connector Picker on the form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
DropdownPrompt		Optional text to show on the drop-down if no connector is selected

Example

```
<bpx:DBConnectorPicker ID="ControlName" DropdownPrompt="Prompt
Text"
    runat="server" />
```

DropDown

This Form control puts a dropdown control on the form.

Properties

PROPERTY NAME	DESCRIPTION
CssClass	To set the CSS class name for this control.
Mode	Used to turn on type-ahead functionality for controls that use Business Values as their data source, by setting the mode to "AutoComplete", e.g. `Mode="AutoComplete"`. With the type-ahead functionality implemented on the control tag, you can link to the Business Value data on the Form definition by: 1. Setting the Dropdown control as an Event Control 2. Mapping the Set Form Data entries to run on the Form Field Event for that Control.

Example

Dropdown with input items manually configured.

```
<bpx:DropDown ID="ControlName" runat="server" >
    <asp:ListItem Text="DisplayText1" Value="Value1" />
    <asp:ListItem Text="DisplayText2" Value="Value2" />
</bpx:DropDown>
```

Dropdown configured for use as a type-ahead dropdown for use with a Business Value.

```
<bpx:DropDown ID="ControlName" Mode="AutoComplete" run-
at="server" ></bpx:DropDown>
```

FormErrorStrings

This Form control is used to identify the area(s) where error messages are displayed on the Form. If this control isn't present on a Form, then the error messages are placed at the top and bottom of the form.

Properties

None

Example

```
<bpx:FormErrorStrings runat="server"/>
```

FormInfoStrings

This Form control is used to identify the area(s) where informational messages are displayed on the Form. If this control isn't present on a Form, then the informational messages are placed at the top and bottom of the form.

Properties

None

Example

```
<bpx:FormInfoStrings runat="server"/>
```

GroupPicker

This Form control will display a Group Picker on the form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
DropdownPrompt		Optional text to show on the drop-down if no user is selected
Height		For ListBox PickerType only; sets the height of the ListBox control
Multiple	True	Allow multiple users to be selected.
PickerType	Dropdown Popup ListBox	Dropdown – use a dropdown control Popup – use a popup control. ListBox – use a ListBox control.
Width		For ListBox PickerType only; sets the width of the ListBox control

Example

```
<bpx:GroupPicker ID="ControlName" DropdownPrompt="Prompt Text"
    PickerType="Popup" runat="server" Multiple="false" />
```

HTML

This Form control will display an HTML string.

Properties

PROPERTY NAME	DESCRIPTION
HTMLString	Enables you to use HTML on a Form. You can use multiple SysVars in this control.

Example

```
<bpx:HTML ID="ControlName" runat="server"
    HTMLString="<a href='{EMAIL_URL}'>click here</a>"/>
```

IgnoreSection

This Form control is used to create a group or section of controls and text on a Form. This section is used to tell the Form processor to ignore the controls inside this section. This can be used, for example, to surround custom controls so that the Form processor doesn't attempt to process the internal form fields.

Properties

None

Example

```
<bpx:IgnoreSection runat="server">
    <!--Items in this section are ignored by the Form pro-
cessor-->
</bpx:IgnoreSection>
```

KView

This Form control is used to place a Knowledge View on the Form. You can use a button to open a Knowledge View or you can view a Knowledge View inline on the form.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Height	Only for iframe – The height of the iframe.	300px
QS	Optional list of QueryString parameters to pass to the KView. The KView can, for example, use these QueryString parameters in its filters. Ensure that you've created a filter corresponding to each QS filter. You must use the QueryString type on the right side in the Knowledge View filter.	
Text	Only for Popup – The text on the button.	View Knowledge View
Type	Iframe – Displays the KView in an	Iframe

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
	inline IFRAME on the form. Popup – A button will be shown on the form. When clicked, a pop window showing the KView will be launched	
Width	Only for iframe – The width of the iframe	100%

Example

```
{KView:ControlName, Text="Text to Display", Type=Popup,
QS="Query String 1",
    QS="Query String 2"}
```

The syntax for the Form Builder is:

```
store={form: store}
appname={form: appname}
```

ListBox

This Form control is used to place a List Box control on a Form, allowing a user to select more than one entry in the list. This ListBox control can be populated via Custom Tasks, scripts, or with the asp:ListItem tag. Note that commas in a ListBox item value aren't valid. For any ListBox item with a comma in the value, the comma will become a semi-colon upon ListBox creation (this doesn't apply when programmatically adding items).

Properties

PROPERTY NAME	DESCRIPTION
Height	(optional) The height of the ListBox control (e.g., 100px, 15ex, 25%, etc.)
Items	The collection of items in the ListBox - See ASP ListControlItems property for usage examples.
SelectedValues	A list of selected values in the Items collection
SelectedValuesString	A comma-separated string representation of the list of

PROPERTY NAME	DESCRIPTION
	selected values in the Items collection
Width	(optional) The width of the ListBox control (e.g., 200px, 20em, 30%, etc.)

Example

```
<bpx:ListBox ID="ControlName" runat="server">
    <asp:ListItem Value="Value1" Text="DisplayText1" />
    <asp:ListItem Value="Value2" Text="DisplayText2" />
    <asp:ListItem Value="Value3" Text="DisplayText3" />
</bpx:ListBox>
```

Print

This Form control puts a single print button control on the form.

Properties

PROPERTY NAME	DESCRIPTION
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.
ImageURL	Sets an optional image for the button.
Style	To set the style for this control.
Text	Displays the label of the button.

Example

```
<bpx:Print ID="ControlName" Text="Text to Display" runat="server" />
```

Radio

This Form control puts a Radio Group control on the form.

Properties

PROPERTY NAME	DESCRIPTION
CssClass	To set the CSS class name for this control.

Example

```
<bpx:Radio ID="ControlName" runat="server" CssClass="MyClass">
    <asp:ListItem Text="DisplayText1" Value="Value1" />
    <asp:ListItem Text="DisplayText2" Value="Value2" />
</bpx:Radio>
```

Rating

This Form control puts a Rating control on the Form.

Properties

PROPERTY NAME	DESCRIPTION
ItemCount	The integer number of stars to display for the rating, up to a maximum of ten.
Precision	Determines whether or not the rating consists of whole stars or some other increment.

Example

```
<bpx:Rating runat="server" ID="ControlName" ItemCount="<int>"
Precision="Item|Half|Exact" />
```

RemoveRow

This Form control will create a button a user can click to remove row(s) to an array.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
ArrayName	The name of the array this button is attached to.	
At	The location to remove the new row(s).	0 (end of array).
ConfirmText	Pops up a confirmation box when a user clicks the button with the specified text, allowing a user to cancel or confirm the action which the button will take.	

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
ImageURL	Sets an optional image for the button.	
OnClientClick	Used to execute client-side JavaScript or call client JavaScript functions. To prevent the button from causing a Post-back, place a return false; at the end of the JavaScript string.	
Rows	The number of Rows to remove.	1
Text	Sets the optional button text.	

Example

```
<bpx:RemoveRow runat="server" ArrayName="MyArray"/>
```

RichText

This Form control places a rich text editor on a Form. This enables a user to enter text as well as format it and place links and other rich text controls within the Form control.

Properties

PROPERTY NAME	DESCRIPTION
Height	(optional) The height of the ListBox control (e.g., 100px)
Width	(optional) The width of the ListBox control (e.g., 200px)

Example

This example will cause a Post-back to the server.

```
<bpx:RichText ID="ControlName" runat="server" Height="NNNpx"
Width="NNNpx" />
```

RoutingSlip

Display the routing slip for the Process Timeline package on the form.

Properties

PROPERTY NAME	DESCRIPTION	PROPERTY ATTRIBUTES	DEFAULT VALUE
ActiveActivityOnly	Should the routing slip only display the active activity?	True False	False
ActiveStepOnly	Should the routing slip only display the active step?	True False	False
ActivityName	An optional comma-separated list of activities to display the routing slip in		
MostRecentInstance	Should the routing slip display only the most recent step instance? If this is false, then the routing slip will show the users every time a step ran.	True False	False
ShowCancelled	Should routing slip display users that	True False	True

PROPERTY NAME	DESCRIPTION	PROPERTY ATTRIBUTES	DEFAULT VALUE
	have been canceled?		
ShowComments	Should Routing Slip display the comments?	True False	True
ShowCompleted	Should routing slip display users that have completed the step?	True False	True
ShowCompletedOn	Shows the date the task was completed for each participant in the routing slip.	True False	False
ShowHeader	Should Routing Slip display the header.	True False	True
ShowInitiator	Should the process initiator be displayed?	True False	false
ShowParticipants	Shows the name of each participant. You can use this, for example, to	True False	True

PROPERTY NAME	DESCRIPTION	PROPERTY ATTRIBUTES	DEFAULT VALUE
	hide the names of the participants, and only show the signature image.		
ShowPending	Should routing slip display users that have not began?	True False	False
ShowReassigned	Should routing slip display users that have been reassigned?	True False	True
ShowResult	Shows the Result column in the routing slip.	True False	True
ShowRunning	Should routing slip display users currently running?	True False	True
ShowSignatures	Should routing slip display user's signatures?	True False	True
ShowStatus	Shows the Status column	True False	True

PROPERTY NAME	DESCRIPTION	PROPERTY ATTRIBUTES	DEFAULT VALUE
	in the routing slip.		
ShowStep	Groups the users in the routing slip according to the step they ran in.	True False	True
ShowTimedOut	Should routing slip display users that have timed out?	True False	True
StepName	An optional comma- sep- arated list of steps to dis- play the rout- ing slip in.		
TimelineName	A partial match for a timeline definition name, restrict- ing the routing slip to be dis- played in the matching Timelines.		
UseDateTime	Optionally shows Date and Time if	True False	False

PROPERTY NAME	DESCRIPTION	PROPERTY ATTRIBUTES	DEFAULT VALUE
	true		
WorkflowName	A partial match for a Workflow definition name, restricting the routing slip to be displayed in the matching Workflows.		

Example

This example will show the routing slip for a single step

```
<bpx:RoutingSlip runat="server" StepName="Step Name"/>
```

This example will show the routing slip for the active step and only show the active users.

```
<bpx:RoutingSlip runat="server" ActiveStepOnly="true"
    ShowRunning="true" ShowCompleted="false"
    ShowCancelled="false" ShowTimedOut="false"/>
```

Save

The "save" (close=false) will only appear when the user is viewing the form in a process task. The "save and close" (close=true) will always appear. The "save and close" will add an entry to the users task list.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Close	The Close parameter can be set to true or false.	True
ConfirmText	Pops up a confirmation box when a user clicks the button with the	

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
	specified text, allowing a user to cancel or confirm the action which the button will take.	
ImageURL	Sets an optional image for the button.	
Text	Sets the optional button text.	

Example

This example will cause a Post-back to the server to save the form contents.

```
<bpx:Save ID="ControlName" runat="server" Text="Text to Dis-
play"
    Close="true"/>
```

Section

This Form control is used to create a group or section of controls and text on a Form. This section can be used to apply formatting, required setting, enabling, or visibility rules to all elements in a section.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
BodyCssClass	To set the CSS class name for the body of the section.	
BodyStyle	To set the style (using any CSS style) for the body of the section.	
CanCollapse	Set this property to "true" males this section a collapsible section.	False
CollapseImageURL	Optional URL to image to use for Collapse	
Expanded	Set to true to have the control viewed in the Expanded state initially. Set to false to have the control viewed collapsed initially.	True

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
ExpandImageURL	Optional URL to image to use for Expand	
HeaderCssClass	To set the CSS class name for the header.	
HeaderStyle	To set the style (using any CSS style) for the header.	
Text	The text for the "header" of the CollapseSection	
WrapperTag	The section can optionally be enclosed with an HTML element such as a div or span. A div uses "block" formatting in HTML (so that the section appears on the next line), where the span uses "inline" formatting.	Div

Example

This sample will create a section on a form to collect addresses. The section will be surrounded with an HTML DIV block.

```
<bpx:Section runat="server" ID="ControlName">
    <!--Some Form controls-->
</bpx:Section>
```

This sample will create a section that flows "inline" with the surround HTML.

```
<bpx:Section runat="server" ID="ControlName"
WrapperTag="span">
    <!--Some more Form controls and HTML-->
</bpx:Section>
```

ShowAttach

This Form control will display a table showing the attachments that match the desired criteria.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
AttachType	Form Workflow Project Process	Attach object(s) directly to Form Attach object(s) to the current Workflow instance as an Workflow reference Attach object(s) to the current timeline instance Attach object(s) to the current Workflow or timeline instance	Process
GroupName		Optional filter of the group that attachments must belong to.	
NameAsView	True False	Enables the attachment name to be a hot link. Same fucntionality as the ShowView property.	False

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
ObjectType	Document Form NotSet	Only shows documents. Only shows form instances Shows all types of objects	NotSet
ShowDate	True False	Show the date for each attachment?	True
ShowDownload	True False	Show the Download link for each document attachment?	True
ShowEdit	True False	Show the Edit link for each document attachment?	False
ShowRemove	True False	Show the Remove link for each attachment?	True
ShowUser	True False	Show the user for each attachment?	True
ShowView	True False	Show the View link for each attachment?	True
Text		Sets the	

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
		optional text to display above attachments.	
ViewInline	True False	Enables the user to select a document to view inline on the Form in an IFRAME.	False
ViewInlineHeight		Optional parameter to set the height of the IFRAME for the selected document. Use any HTML compatible string such as 300px.	200px

Example

```
<bpx:ShowAttach ID="ControlName" runat="server"
GroupName="Group"/>
```

ShowAttachKView

This Form control will add an Iframe to the Form showing a Knowledge View of attached objects.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
AttachType	Form	show object(s)	Process

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
	Workflow Process Timeline Process	attached to Form show object(s) attached to the current Workflow instance as a Workflow reference show object(s) attached to the current Process Timeline instance as a Process Timeline reference show object(s) attached to the current Workflow or Process Timeline instance	
GroupName		Specifies to which group objects shown on the Knowledge View should be restricted.	
Height		Height of the Iframe in	

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION	DEFAULT VALUE
		pixels.	
ShowParents		If set the 1, the Knowledge View will show the parents of the object.	Null (parent isn't shown)
Width		Width of the Iframe in pixels.	

Example

```
<bpx:ShowAttachKView ID="ControlName", Height="NNN",
Width="NNN" />
```

ShowReport

This Form control will add a viewer to a Form to display a Report object in the Form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
Height		The height, in pixels, of the Report's display on the Form.
ID		The Control's ID.
ImageOnly	Boolean	Display an image only for the Report
ImageURL		The IRL of an image to display for the Report.
QS		The query string to use for the report's parameters, if any.

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
RID		The Report's ID.
SmallImage	Boolean	Display the image in a small format.
Text		Text to display as the Report's Title.
Type	IFrame Popup Image HTML	The display type for the control to display on the form.
Width		The Width, in pixels, of the Report's display on the Form.

Example

```
<bpx:ShowReport ID="ControlName" runat="server" RID="ReportID"
Height="NNN" Width="NNN"
    Text="Text to Display" ImageURL="ImgURL" ImageOnly="true"
SmallImage="True" Type="IFrame"
    QS="Query String">
</bpx:ShowReport>
```

SignatureComments

This Form control will display a textbox on the form to enter Workflow comments.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
Columns	Sets the width of the textbox.	70
Rows	Sets the height of the textbox.	4
Style	To set the style for this control.	
Text	Sets the optional default text in the textbox.	

Example

```
<bpx:SignatureComments id="ControlName" runat="server"
    Text="Text to Display"/>
```

Slider

This Form control will display Slider control on the Form to set a value between 0 and 100.

Properties

PROPERTY NAME	DESCRIPTION	DEFAULT VALUE
TrackPosition	The visual position of the track indicator.	BottomRight
Height	Sets the height of the control. (Optional)	
Width	Sets the width of the control. (Optional)	
ShowDecreaseHandler	Displays the icon to decrease the slider value. (Optional)	True
ShowIncreaseHandle	Displays the icon to increase the slider value. (Optional)	True
WrapperTag	The outer HTML tag, e.g. "DIV" to use as a container for the control, if any. (Optional)	
Alt	The Alt Text to use for access-ibility.	

Example

```
<bpx:Slider runat="server" TrackPosition="Bot-
tomRight|Center|TopLeft" ID="ControlName"
    Style="CSSStyles" CssClass="ClassName" Height="<int>"
Width="<int>"
    ShowDecreaseHandler="true|false" ShowIncreaseHandle="true|-
false"
    WrapperTag="HTMLTag" Alt="Alt Text" />
```

Sort

This Form control will place a button on a Form which will sort an array.

Properties

PROPERTY NAME	DESCRIPTION
ArrayName	Specify the name of the array to sort.
Descending	Specify true to sort in descending order.
ImageURL	Sets an optional image for the button.
Primary	Specify the column name of the primary sort key.
Secondary	Specify the column name of the optional secondary sort key.
Tertiary	Specify the column name of the optional tertiary sort key.
Text	Sets the optional button text.

Examples

```
<bpx:Sort runat="server" ID="ControlName" Text="Text to Dis-
play"
     ArrayName="MyArray" Primary="ColumnName" />
```

Sum

This Form control sums all items of a column in an array. Please note the ID is optional as it can be a system variable.

Properties

PROPERTY NAME	DESCRIPTION
Column	The column in an array that you'd like to sum up.

Example

```
<bpx:Sum ID="ControlName" runat="server" Column="ColumnName"/>
```

SysVar

This Form control is used to evaluate and display a System Variable. You can pass an entire System Variable string in the SysVarString property, or you can break apart the System Variable into individual properties.

Properties

PROPERTY NAME	DESCRIPTION
	-Or-
Encode	Optionally set the encoding type.
IfNull	Optionally set the "If sysvar is null" string.
Post	Optionally set the "postfix" string.
Pre	Optionally set the "prefix" string.
SysVarName	Set the system Variable name.
SysVarNamedParms	Optionally sets the System Variable named parameters. Multiple parameters can be passed separated by a comma.
SysVarParms	Optionally sets the System Variable parameters. Multiple parameters can be passed separated by a colon.
SysVarString	Set this property to pass an entire System Variable in one string.

Example

```
<bpx:SysVar ID="ControlName" runat="server"
SysVarName="SYSVAR_NAME"
    SysVarNamedParms="SysVarParam1:SysvarParam2"/>
```

Use Case Examples

This example displays the date 1 week from now.

```
<bpx:SysVar ID="SysVar1" runat="server" SysVarName="CURR_DATE"
    SysVarNamedParms="Days=7"/>
```

This example displays the current user's ID on the form. Notice the ! to HTML-encode the string.

```
<bpx:SysVar ID="SysVar2" runat="server" SysVarString="{!CURR_
USER }"/>
```

Tab

This control is used to identify each tab in a tab strip. The value of the TabStrip will be the selected Tab.

Properties

PROPERTY NAME	DESCRIPTION
PageViewID	The ID of the related TabContent control
Text	Optional text for the actual Tab
Value	Optional value for the control if this tab is selected

See the TabContent control below for an extended code sample.

TabContent

This Form control contains the actual content for a single tab. You can place any number of Form controls inside a TabContent section.

Properties

PROPERTY NAME	DESCRIPTION
ID	The ID of the content for the tab strip

Example

```
<bpx:TabStrip runat="server" ID="MyTabStrip"
    MultiPageID="MyTabStrip_Content">
    <Tabs>
        <bpx:Tab runat="server" PageViewID="tab1" />
        <bpx:Tab runat="server" PageViewID="tab2" />
    </Tabs>
</bpx:TabStrip>
<bpx:TabStripContent runat="server" ID="MyTabStrip_Content">
    <bpx:TabContent ID="tab1" runat="server">
        <!--Form data inside tab 1. This can include any form
control-->
    </bpx:TabContent>
    <bpx:TabContent ID="tab2" runat="server">
        <!--Form data inside tab 2-->
    </bpx:TabContent>
</bpx:TabStripContent>
```

TabStrip

This Form control creates a tab strip section.

Properties

PROPERTY NAME	DESCRIPTION
ID	Optional ID if the value of the selected tab should be maintained
MultiPageID	The ID of the related TabStripContent control

See the TabContent control above for an extended code sample.

TabStripContent

This control identifies the actual contents of the various tabs.
Properties

PROPERTY NAME	DESCRIPTION
ID	The ID of the content for the tab strip

See the TabContent control above for an extended code sample.

TimePicker

This Form control places a picker for selecting time values on a Form.
Properties

PROPERTY NAME	DESCRIPTION
EndTime	(Optional) Sets the maximum time (of day) for the pre-selected picker values. Must exceed the StartTime value.
Interval	(Optional) The amount of time (in minutes) between pre-selected picker values.
StartTime	(Optional) Sets the beginning time (of day) for the pre-selected values available for the picker.

Example

```
<bpx:TimePicker ID="ControlName" runat="server"
StartTime="11AM"
    EndTime="3:45pm" Interval="15" />
```

UserPicker

This Form control will display a User Picker on the form.

Properties

PROPERTY NAME	PROPERTY ATTRIBUTES	DESCRIPTION
DropdownPrompt		Optional text to show on the drop-down if no user is selected
Height		For ListBox PickerType only; sets the height of the ListBox control
InGroup		Optional filter to only show users that are members of the specified group.
Multiple	True	Allow multiple users to be selected.
PickerType	Dropdown Popup ListBox	Dropdown – use a dropdown control Popup – use a popup control. ListBox – use a listbox control.
Width		For ListBox PickerType only; sets the width of the ListBox control

Example

```
<bpx:UserPicker ID="ControlName" DropdownPrompt="Prompt Text"
    PickerType="Popup" runat="server"
    Multiple="true" InGroup="GroupName" />
```

Form Controls and JavaScript

You can use JavaScript to access the actual HTML controls for BP Logix controls and bind JavaScript events to them. For example, the JavaScript:

`CurrentForm.FormControls["YOUR_CONTROL"]`

returns the HTML object in JavaScript. Therefore, you could use:

`CurrentForm.FormControls ["BpTextBox1"].onkeypress = BpTextBox2_ Keypressed;`

to bind to the **onkeypress** event.

Note that if you want this event to fire even after a postback (such as a button event), you should place this into a bpUserJavaScript function on your page. This function will be called after the initial page load, AND after each postback. For example, place this on your ASCX page:

```
function bpUserJavaScript()
{
    CurrentForm.FormControls["ControlName"].onkeypress = BpTex-
tBox2_Keypressed;
}
```

In addition, you can simply use the native ASP.NET textbox, checkbox, radio button, or dropdown controls and their corresponding server tags, and use standard JavaScript for those controls. Other native ASP.NET controls aren't supported for this functionality.

Custom Workspace Portlets

When creating home page portlets in workspaces, it's often helpful to use a particular URL to display in a portlet, using the "Specific URL" setting in the portlet's configuration. For internal URLS, however, the relative paths may change to the URL depending on whether the workspace is on a development server or a production server. You may also want to show different URLs in a portlet based on the user's identity, work center, etc. In cases like these, you need to be able to make the URL dynamic, rather than using a hard-coded URL. Through the use of Custom Variables you can create a variable to store the appropriate URLs you might wish to display, as well as add logic to determine when different URLs should be displayed.

Process Director allows you to set your own system variables in the vars.cs.ascx file. All custom vars that you create are formatted as strings, so you should use string syntax when setting the variable's value, i.e., use double quotes ("") around the value. Process Director will convert custom vars to numbers on the fly if numeric comparisons or calculations are required. Custom vars that you create are stored in a custom dictionary as key/value pairs.

There are two methods available in the custom vars file for creating custom vars: SetSystemVars and PreSetSystemVars. The PreSetSystemVars method is called prior to initializing the Process Director database, while the SetSystemVars method is called after database initialization.

In general, this means that the default method to use when creating custom vars is the PreSetSystemVars method; however, if you need to set the value of the var based on information stored in the database, such as the identity of the user or the workspace in which the user is working, you must use the SetSystemVars method.

The following example shows how to set a simple custom variable to a URL.

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //This creates a new custom var that can be access from
anywhere within
    //Process Director by using the syntax {CustomVar:PORTLET_
URL}
    bp.Vars["PORTLET_URL"] = "http://www.myurl.com";
}
```

You can also add any programming logic you'd like here, to display different URLs under the different conditions you devise.

Using the syntax `{CustomVar:MY_VAR}` elsewhere in Process Director will return the custom variable (in this case, "MY_VAR") value that was defined in the vars.cs.ascx file.

In the Home Page Windows tab of the Workspace configuration screen, set the Type of the portlet in which you wish to display a URL to "Specific URL". An input box labeled "URL" will appear beside the Type dropdown.

In the URL input box, type the name of the custom variable you created, using the following syntax:

`{customvar:PORTLET_URL}`

Custom Tasks

Custom Tasks can be written in Process Director and used by Forms and Process Timeline definitions. A Custom Task enables common business logic to be packaged in a reusable way and made available to users that are building Forms and Process Timelines. You should be familiar with writing custom scripts for Forms and Process Timelines before writing a Custom Task. There are two types of Custom Tasks; Form Custom Tasks and Process Custom Tasks.

What Custom Tasks Can Be Used For

Custom Tasks can provide various types of business functions that are unique to your environment, or common functions that can be used across multiple Form and Process Timeline definitions. For sample Custom Tasks contact BP Logix.

Form Custom Tasks

A Form Custom Task provides users that are building Forms the ability to execute custom business logic when the end-user is filling out the Form. A Custom Task is mapped to an event on the Form (e.g., a button). When the end-user triggers this event, the Form Custom Task script will be executed. A Form Custom Task can provide a mapping of external fields (e.g. database) to the fields on a Form.

When the script for a Form Custom Task is run, it can optionally display an interface to the user. The Custom Task script can query and/or set any of the form data on the container Form.

Process Custom Tasks

A Process Custom Task provides users that are building processes the ability to use custom business logic in the Process Timeline definition. The Custom Tasks will display a Form interface to the user building the Process Timeline that enables them to configure the data required for the Custom Task. This configuration form is available from the Custom Task tab of a Process Timeline activity of the Custom Task Activity Type. This configuration data will be available to the Custom Task script when the Timeline Activity is run. The Custom Task script will run in a similar environment to a Timeline script and has access to all Timeline objects and their data.

A Process Custom Task can't display an interface to the end-user participating in the process. The only interface displayed is during the configuration of the Custom Task in the Process Timeline definition.

How Custom Tasks Work

Custom Tasks in Process Director provide a way to package reusable functions for Form builders and process implementers. They consist of a Form and a script file. This enables your business logic to be placed in the script, and the Form is used to collect configuration information for the script, and optionally can display an interface (Form Custom Tasks only). a Form Custom Task can be mapped to different

events (e.g. form display, button, etc.) on a standard Form and are run when the event occurs.

Configuration vs. Running

The Custom Tasks are executed in the following modes:

ConfigurationThis is when the user is configuring the data needed for the Custom Task to run. During the configuration, it is common to display a list of the form fields in the container Form. This enables fields to be mapped to external sources by the builder of a Form, without requiring them to have any special naming convention for their form fields.

Run-time (run)The Custom Task is run when a mapped Form event is triggered or when aCustom Task Timeline Activity has been invoked. When a Custom Task is run, it has access to the configuration data and the container Form.

Configuration Data

When a Custom Task is configured, a form data instance is created that contains the configuration data entered. This form instance is stored "under" the Form definition or Process Timeline definition, depending on the type of Custom Task. When the Custom Task is run (whether in a Form or Process Timeline), it has access to all this configuration data in this form instance.

Container Form

A Custom Task can have a "container Form". This is the standard Form that is displayed to the user. a Form Custom Task is mapped to an event on the container Form. A container Form is optional for a Process Custom Task, but can be specified as part of the step configuration on the Custom Task tab of the Timeline Activity's definition. The Form and script part of a Custom Tasks has access to the container form data.

Creating a Custom Task

A Custom Task requires a Form created in the .NET environment similar to how a Form is created. To create a Form for a Custom Task, use the Create New menu item in the Content List and select Form Definition in the dropdown. Then select from the template dropdown "Use Empty .ASCX form". The Form must be created using the special options in the Form Definition page that is displayed. A Custom Task can be created as a Form-only Custom Task, a process-only Custom Task, or a

Custom Task that can be used by both Forms and processes. This Form will also display any Custom Task configuration.

These events are called for forms and while configuring a Custom Task:

```
public override void BP_Init()
public override void BP_Event(bp.EventType pEventType, string
pEventName)
public override void BP_Validation()
public override void BP_Display()
public override void BP_Completed()
```

When a Custom Task is mapped to a Form, the following Custom Task function will be called:

```
protected override bool CT_RunTask(bp.FormCTMappedTo
pMappedTo, string pEventName)
```

This event is called to "run" a Custom Task. It should return "true" to show the Custom Task GUI, or false to continue processing the form.

```
public override bool CT_RunTask(bp.FormCTMappedTo pMappedTo,
string pEventName)
```

This function should return "true" to show the Custom Task GUI.

This is created like any other Custom Task (Form, Process), and when the rule is evaluated that uses the CT, this event will be called:

```
public override DataItem CT_RunRuleTask()
```

This is so the Custom Task can return a string, a list of UIDs, etc.

These events are called for Custom Tasks while running a GUI:

```
public override void CT_Init()
public override void CT_Event(bp.EventType pEventType, string
pEventName)
public override bool CT_Validation()
public override void CT_Display()
public override void CT_Completed()
```

Use this.FormInfo to access the current Form info. For example:

```
this.FormInfo.FormControl("SomeTextControl").Text = "This is
new text";
```

While in a Custom Task run or GUI, use this.ContainerFormInfo to access the containers form info (get/set form fields, etc). For example:

```
this.ContainerFormInfo.FormControl("SomeContainerControl").Re-
quired = true;
```

Use this.FormInfo.AddInfoMessage and this.FormInfo.AddErrorMessage to add error or info messages to forms (e.g., for validation). For example:

```
this.FormInfo.AddInfoMessage(new FormMessageString("This info
message
    will display on the form!"));
```

A Custom Task may have 2 sections, FormControlCustomTaskConfig and FormControlCustomTaskRun. For example:

```
<bpx:FormControlCustomTaskConfig runat="server">
<!--This section includes stuff that is shown when a Custom
Task is
being configured-->
    <asp:TextBox ID="config_text1" run-
at="server"></asp:TextBox>
</bpx:FormControlCustomTaskConfig>

<bpx:FormControlCustomTaskRun runat="server">
    <!--This section includes stuff that is shown when a Cus-
tom Task is
    run inside a Form (if CT_RunTask returns true).-->
    <asp:TextBox ID="run_text1" runat="server"></asp:TextBox>
</bpx:FormControlCustomTaskRun>
```

Web Service Custom Tasks

Some of the Custom Tasks can utilize Web Services (e.g. calling a remote Web Service to look up employee information from an employee ID stored on a form). Only certain types of Web Services are directly supported. These Web Services must use primitive data types such as numbers and strings as parameters. Other, more complex Web Services, can be supported by "wrapping" them in a proxy Web Service. This proxy Web Service calls the actual web service, but exposes it using primitive data types that the Custom Tasks support.

An example of a Web Service wrapper is installed in the "C:\Program Files\BP Logix\Process Director" folder as bpProxy.zip. This file contains a complete Visual Studio project creating both a .ASMX Web Service, and a .DLL that has the web

reference to the complex Web Service. This Web Service wrapper can be deployed on any IIS server. It can also easily be deployed in the Process Director web site by copying the .DLL to the "C:\Program Files\BP Logix\Process Director\website\bin" folder, and the .ASMX to the "C:\Program Files\BP Logix\Process Director\website\custom\services" folder. After they are deployed, then can be reference like any other web service.

For more information, refer to the Web Services topic.

Classes

Custom programming in Process Director is performed by using a number of classes that are built into the product SDK. Each class in the Process Director library contains all of the relevant Properties Methods and Events, along with the required and optional parameters available for each method. Each of the classes are listed in the Table of Contents displayed in the upper right corner of the page.

In most cases, each class is named after the specific Process Director object the class addresses. For instance, the **ProcessTaskUser** class provides the Methods for manipulating the user assigned to the generic process task that's created by a running Process Timeline Activity. Similarly, the **ProjectActivityUser** class contains the Methods for manipulating the user assigned to a specific Process Timeline Activity.

Exceptions to this Process Director object-based naming convention include:

- The **bp** class, which is the base class/namespace for all Process Director object classes. This class is universal, and the global bp variable provides the system context that is passed to many functions.
- The **PDF** class, which provides the methods for manipulating PDF documents, irrespective of the Process Director object to which the PDF file is associated.
- The **Excel** class, which provides the methods for manipulating Microsoft Excel documents, again, irrespective of the Process Director object to which the file is associated.

Common Termination Reasons

The classes associated with Timeline Activities, Process Tasks, and Workflow Steps all have a common set of termination reasons associated with every task or activity. The termination reason is set when the task or activity is completed. A list of those classes would include:

- ProcessTask
- ProcessTaskUser
- ProjectActivity
- ProjectActivityUser
- Task
- WorkflowStep
- WorkflowStepUser

In the Process Director UI, text values are returned for the termination reasons. In Process Director database tables, however, numeric values are stored for the termination reason in the nTermReason field of the appropriate table. Thus, any records returned by database queries or views will display the numeric value for the termination reason. The numeric values that may appear in the nTermReason field, along with a description of each value, is provided in the Enumerators section of Database Guide's Table Definitions topic.

bp Class

This object represents the SDK environment. All scripts will have access to this object. This class contains many methods that implement utility routines (such as logging, data conversion, etc).

Methods

CheckForAdvance

This API will force the process engine to start the task advance check logic.

Parameters

WFINSTID: Optional Workflow Instance ID to check.

PRINSTID: Optional Timeline Instance ID to check.

WFID: Optional Workflow Definition ID to check.

PRID: Optional Timeline Definition ID to check.

Returns

None.

Example

```
bp.CheckForAdvance(ProcessInstanceID, null, null, null);
```

DateDiff (Static Method)

This API will determine the value of the difference between two dates.

Parameters

BP: The bp environment.

DateFrom: The origin (starting) date

DateTo: The termination (ending) date

DifferenceType: The type or units of difference to calculate (e.g. days, seconds, hours, years)

Returns

int: An integer representing the value of the difference between the two dates in the specified units.

Example

```
// Should return the value '4'
int days = bp.DateDiff(bp, DateTime.Parse("2022-03-05"),
    DateTime.Parse("2022-03-09"),
    BPLogix.WorkflowDirector.SDK.FormControls.DateDiffType.Day-
s);
```

DBOpenComplete

This boolean property returns the status of the Process Director Database.

This can be used, for example, prior to making SDK calls that may access the database in the custom vars file.

Parameters

Returns

boolean: This property returns true if the Process Director database has been successfully opened.

Example

```
// Called after database initialized
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Before we make SDK calls that access the database,
    // ensure DB has been opened
    if (bp.DBOpenComplete)
    {
        // Routine to perform if true
    }
}
```

FormatCurr (Static Method)

This API will return a string that is formatted as a currency.

Parameters

Any integer, double, decimal, or string

Returns

string: The object formatted in a currency (depending on locale)

Example

```
string TempString;

// Returns $100.00 (assuming USA locale)
TempString = bp.FormatCurr("100");

// Returns $10.10 (assuming USA locale)
TempString = bp.FormatCurr(10.1);

// Returns ""
TempString = bp.FormatCurr("Bad Value");
```

GetExcelRows

This API imports an Excel file into memory.

Parameters

ExcelPath: The server local path to the Excel file to import (specify this or ExcelDID).

ExcelDID: The ID of the document in the repository of the Excel file to import (specify this or ExcelPath).

SheetNumber: Zero-based sheet number to import (specify this or SheetName).

SheetName: Sheet name to import (specify this or SheetNumber).

Returns

list<bpExcelRow>: List of excel rows, or null if any error

Example

```
// This will import an Excel file
var Rows = bp.GetExcelRows( "C:\\db_import.xlsx", // The file
path to import
                            null,  // null since we are using
a file path
                            0,  // The first sheet in the work-
book
                            null);  // Null since we are using
SheetNumber
                                    // instead of SheetName

    if (Rows != null)
    {
        foreach (var Row in Rows)
        {
            foreach (var Col in Row.Columns)
            {
                bp.log0("Column value: " + Col.Value + ",
Column type: "
                + Col.Type);
            }
        }
    }
```

GetTempDirectory

This API will return a string for a local temporary directory.

Returns

string: The name of a local file system temporary directory.

GetTempFilePath

This API will return a string for a temporary file name. It will use random digits to ensure the name is unique.

Parameters

An optional string to prepend, and optional string to append

Returns

string: The string of a unique temporary file name and path.

Example

```
var TempFileName = bp.GetTempFilePath("begin","end");
```

GetCurrentProfileName

This API will return the string name of the currently selected profile for the current user. If there is no profile selected, an empty string is returned.

Returns

string: The name of the current profile.

Example

```
bp.msg0("Current profile name: " + bp.GetCurrentProfileName());
```

HTMLEncode / HTMLDecode (Static Method)

These APIs will HTML-encode or HTML-decode a string.

Parameters

string: the value to convert

Returns

string: Appropriate encoded or decoded value

Example

```
// Returns "You &amp; me"
string test1 = bp.HTMLEncode("You & me");

// Returns "You & me"
string test2 = bp.HTMLDecode("You &amp; me");
```

HTTPRequest (Static Method)

This API will return the content of a specified web page.

Parameters

String URL: the URL to get

Returns

string: The HTML content of the web page.

Example

```
// This will load the web page from google.com into a string
variable
string Result = bp.HTTPRequest("http://google.com");
```

HTTPRequestBytes (Static Method)

This API will return the bytes of a URL.

Parameters

String URL: the URL to get

Returns

byte array: URL contents

Example

```
// This will load the web page from google.com into a byte
array variable
var Result = bp.HTTPRequestBytes("http://google.com");
```

HTTPPoke (Static Method)

This API will POST a URL.

Parameters

String URL: the URL to POST

Returns

N/A

Example

```
// This will POST a certain web page
bp.HTTPPoke("http://myserver.com/pagetopost.aspx");
```

HTTPRest (Static Method)

This API will POST a URL and return the XML response as a string

Parameters

string (URL): the URL to POST

string (XML Data): request data to POST (typically XML)

Returns

string: String variable containing response from page – typically in XML.

Example

```
// This will POST a request and get back the response
bp.HTTPRest
("http://myserver.com/pagetopost.aspx","<API>1</API>");
```

ImportExcelDatabase (Static Method)

This API import an Excel file into a database. The sheet name will be used as the table name in the database. Only sheets with data in cell A1 will be imported. Row 1 in Excel is used to specify the column name in the database. You can optionally specify a column type for each column by separating the column name from the type with a colon. You can use several database independent column types, such as"

- BP_STRING
- BP_DECIMAL
- BP_BOOL
- BP_INT
- BP_DATETIME

If you don't specify a type, the import process will guess at the type by looking at the first row of data. Since the worksheet must have a header row to supply column names and optional data types, the first data row will be row 2 of the work-sheet.

Parameters

BP: The bp environment.

DestDB: The DataSource of the database to insert the Excel data.

ExcelPath: The server local path to the Excel file to import (specify this or ExcelDID).

ExcelDID: The ID of the document in the repository of the Excel file to import (specify this or ExcelPath).

TBLPrefix: Optional prefix to add to all imported tables.

DropFirst: Should the tables be dropped before importing?

DoCreate: Should the tables be created during the import?

DeleteFirst: Should the rows be deleted before the import?

Returns

Boolean: True if the operation succeeds.

Example

```
// This will import an Excel file into the database
bp.ImportExcelDatabase(bp,
    dbConnection, // The database to insert the Excel data
    "C:\\db_import.xlsx", // The file path to import
    null, // null since we are using a file path
    "USER_", // All tables created will have USER_ prefix
    true, // drop existing tables first
    true, // create all tables in database
    false); // no need to delete rows –
                        // entire tables were dropped
```

INT | CURR | DOUBLE | DECIMAL | BPDATETIME (Static Method)

These APIs will convert a string into the appropriate data type.

Parameters

string: the value to convert

Returns

Appropriate data type depending on method.

Example

```
// returns 10
int TempInt = bp.Int("10");

// returns DateTime object
DateTime TempDate = bp.bpDateTime("1/1/2010");
```

ImportXML, ImportGlobalKViewXML, ImportProfilesXML

These functions import XML data from an XML file that Process Director exported. They mainly take the same parameters, but interpret the XML in different ways:

- ImportXML: reads the XML data into a file in the content list.
- ImportGlobalKViewXML: reads the XML file into a global Knowledge View.
- ImportProfilesXML: reads the XML file into a profile definition.

The method declarations are as follows:

```
public bool ImportXML(string XMLPath,
                      string XMLDID,
                      byte[] XMLData,
                      string PID,
                      string ParentFolderID,
                      out List<string> RetMsg)


public bool ImportGlobalKViewXML(string XMLPath,
                                 string XMLDID,
                                 byte[] XMLData,
                                 out List<string> RetMsg)


public bool ImportProfilesXML(string XMLPath,
                              string XMLDID,
                              byte[] XMLData,
                              out List<string> RetMsg)
```

Parameters

string XMLPath: The file path of the XML file.

string XMLDID: The document ID of the XML file.

byte []XMLData: The raw XML data.

string PID: ID of the partition in which to import this XML document.

string ParentFolderID: The ID of the folder that this XML document will be imported into.

out List<string> RetMsg: A list of messages returned by this function.

Returns

boolean: This function returns true if, and only if, the import is successful.

Example

```
//imports XML File
List<string> returnMessages = new List<string>();
bool success = bp.ImportXML("someXMLFile.xml", "1234", "1234",
    returnMessages);
```

log0 | log1 | log2 | log3 | log4 | log5 (Static Method)

These APIs will send a logging message to the bp.log file. This file can be viewed on disk, or you can use the web based log viewer. log0 will always be sent to the log file. The other methods will be sent to log file depending on the system log level setting.

The actual log will be written conditionally depending on the current logging level of the system. log0 will always be written, log1 will only be written if the logging level is 1 or higher, etc.

More information about logging can be found in the System Administrator's Guide.

Parameters

Any formatted string or string variable.

Returns

None

Example

```
bp.log0("This is a test");
bp.log0("Some sample data " + CurrentForm.FormControl
("mydata").Text;
bp.log0("Data1: {0} Data2: {1}", 10, 20);
```

Login (Static Method)

This API will login as the specified user.

Parameters

BP: The bp environment.

User: The UserID to login.

Password: The password to login.

Returns

boolean: True if the operation succeeds.

Example

```
// This will login the user from the edit fields on a form
bp.Login(bp, CurrentForm.FormControl("userid"),
    CurrentForm.FormControl("password"));
```

RunKView

This API runs a Knowledge View and returns the found rows of Content items.

Parameters

KVID: The Knowledge View ID to run.

Filter: List<NameValue> - Optional list of input filter to the Knowledge View. Use {var:PARM} in the Knowledge View definition to access the filter fields.

StartFID: Optional Folder ID to filter.

StartCATID: Optional Category ID to filter.

Output: List<KViewResult> - List of resultant rows

Returns

boolean: True if the operation succeeds.

Example

In order for the RunKView method to execute properly, the Knowledge View defin-ition must have the appropriate system variable set as a filter. In the case of the example below, the Knowledge View must use the value of "Filter1" set as a filter. For instance, let's say you wish to find the value of a filter named "Filter1" in a Form field. In that case, your filter in the Knowledge View definition might look something like this:

The RunKView method will pass the filter value in the Filter parameter, which is identified by the "infilter" variable in the code snippet below.

```
// Run a Knowledge View, then log the results
var infilter = new List<NameValue>();
infilter.Add(new NameValue("Filter1","Value"));
List<bp.KViewResult> rows;
```

```
var res = bp.RunKView(CurrentForm.FormControl("kview_
picker").Text, infilter,
    null, null, out rows);
if (res)
{
    foreach (var row in rows)
    {
        bp.log0("ID: " + row.ID + " type: " + row.ObjectType);
        foreach (var col in row.Columns)
        {
            bp.log0("Column Name: " + col.Name + " Value: " +
col.Value);
        }
    }
}
```

SendEmail (Static Method)

This API will send an email. This is an overloaded method, and the method declarations are as follows:

```
public      static      bool      SendEmail      (bp           BP,
                    string                     pFORMID,
                    string                   pFORMINSTID,
                    List<ContentObject>     pAttachments,
                    string                     pSubject,
                    string             pToEmailUID,
                    string pFromEmailAddress
```

```
public      static      bool      SendEmail      (bp           BP,
                    string                     pFORMID,
                    string                   pFORMINSTID,
                    List<ContentObject>     pAttachments,
                    string                     pSubject,
                    string             pToEmailAddress,
                    string             pToEmailDisplay,
                    string pFromEmailAddress)
```

Parameters

BP: The bp environment.

FORMID: The form ID to use as the email template.

FORMINSTID: The ID of the form instance to be used for variable substation in the email. Specify "null" if you don't wish to reference a form instance.

Attachments: A list of ContentObject attachments to add to the email

Subject: The subject of the outgoing email

ToEmailUID: The UID of the destination

FormEmailAddress: The email address of the sender of the email

Returns

boolean: True if the operation succeeds.

Example

```
bp.SendEmail(bp,
    oMyFORM.ID,              // The email template
    oMyFORMINST.ID,          // The form instance
    null,                    // list of attachments
    "My Email Subject",      // Subject
    newuser.UID,             // The ID of the "to" for the
email
    "from_email@company.com"); // The "from" email address/
```

Business Value Class

This object represents a Business Value object.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
ID	String	The ID of the Business Value
Name	String	The name of the Business Value
Text	String	The Text of a Business Value property
Value	Data type specified in the data source	The Value of a Business Value property
Values	List object	A list object containing all of the Values returned by a Business Value
Number	Decimal	If a Business Value property returns a numerical value, the Number property

PROPERTY NAME	DATA TYPE	DESCRIPTION
		casts the value as a decimal number. If the value isn't a number, the Number's property will return "0".
Numbers	Decimal	If a Business Value property returns a numerical value, Numbers property will return a list object containing all of the values, cast as decimals numbers. If the the value isn't a number, the Number's property will return "0".
DateTime	DateTime	If a Business Value property returns a DateTime value the DateTime property will return the value cast as a DateTime object.
DateTimes	List Object	If a Business Value property returns a DateTime value the DateTimes property will return a list object with all the values cast as a DateTime object.

Methods

GetBusinessValueByID

This API will return the Business Value object specified by the Business Value ID.

Parameters

bp: The Process Director environment.

pBVID: The string ID of the Business Value.

Returns

object: A Business Value object.

Example

```
var myBV = BusinessValue.GetBusinessValueByID(bp, "Busi-
nessValueID");
```

GetBusinessValueByName

This API will return the Business Value object specified by the Business Value's Name.

Parameters

bp: The Process Director environment.

PID: The string PartitionID of the partition in which the Business Value is located.

pName: The name of the Business Value.

Optional Parameters

pGroup: The Group Name configured for the Business Value. This is the group name specified in the Group in Value Picker property on the Business Value definition's Options tab.

Returns

object: A Business Value object.

Example

```
var myBV = BusinessValue.GetBusinessValueByName(bp, "Par-
titionID", "BVName");
```

Parameters

This API will return a list containing all of the names of the Parameters for a Business Value.

Parameters

None

Returns

list <string>: List of parameters.

Example

```
var myBV = BusinessValue.GetBusinessValueByID(bp, "Busi-
nessValueID");
var myParams = myBV.Parameters();
```

Properties

This API will return a list containing all the properties of a Business Value.

Parameters

None

Returns

list <object>: List of BusinessValueProperty objects.

Example

```
var myBV = BusinessValue.GetBusinessValueByID(bp, "Busi-
nessValueID");
var myProps = myBV.Properties();
```

Property

This API will return a specified Business Value property.

Parameters

pPropName: The string name of the Property to return.

Returns

object: A Business Value Property object.

Example

```
var myBV = BusinessValue.GetBusinessValueByID(bp, "Busi-
nessValueID");
BusinessValueProperty myProp = myBV.Property("PropertyName");
string val = myProp.Value;
```

SetParameter

This API will set a Parameter value for a Business Value.

Parameters

pParamName: The string name of the Parameter to set.

pParamValue: The string value to set for the Parameter.

Returns

None

Example

```
var myBV = BusinessValue.GetBusinessValueByID(bp, "Busi-
nessValueID");
myBV.SetParameter("ParamName", "ParamValue");
```

TextList

This API will return a list containing all of the Display Strings for a Business Value Property, as opposed to the Property values, those these will usually be the same. This is most often useful for filling dropdown controls that use different display and value fields.

Parameters

None

Returns

list <string>: List of Display Strings.

Example

```
var myBV = BusinessValue.GetBusinessValueByID(bp, "Busi-
nessValueID");
BusinessValueProperty myProp = myBV.Property("PropertyName");
List<String> val = myProp.TextList;
```

Case Class

This class represents a Case object.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Group	String	The Group Name configured for the Case.

Methods

AddToCase

This API will add a content object to a case instance.

Parameters

pObj: The content object to add to the case.

Returns

boolean: Returns"true" if the operation succeeds

Example

```
var newCase = Case.CreateCase(bp, "CASEID");
newCase.AddToCase(CurrentForm);
```

CreateCase

This API will create a new instance of a Case object.

Parameters

bp: The Process Director environment.

pCASEID: The CaseID of the Case definition.

Returns

object: A Case Object.

Example

```
var newCase = Case.CreateCase(bp, "CASEID");
```

GetCaseByCASEID

This API will return a specified Case object.

Parameters

bp: The Process Director environment.

pCASEID: The CaseID of the Case definition.

Returns

object: A Case Object.

Example

```
var myCase = Case.GetCaseByCASEID(bp, "CASEID");
```

GetCaseByCASEINSTID

This API will return a specified Case object.

Parameters

bp: The Process Director environment.

pCASEINSTID: The CaseInstanceID of the Case instance.

Returns

object: A Case Object.

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
```

GetCaseProperties

This API will return a List object containing all of the properties for a Case.

Parameters

None

Returns

list <object>: A list of Case property objects.

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
List<NameValueEx> myVals = myCase.GetCaseProperties();
// Get the first value from the list
string val = myVals[0].Value;
```

GetPropertyText

This API will return the specified property of a Case object. This API will return the value from any case property type, including number or DateTime values, cast as a string.

Parameters

pPropName: The string name of the property to return.

Returns

string: The case property value.

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
string myVal = myCase.GetPropertyText("PropertyName");
```

GetPropertyNumber

This API will return the specified property of a Case object, if the property is a numeric value. If the property is a string or a DateTime, the API will return "0".

Parameters

pPropName: The string name of the property to return.

Returns

decimal: A decimal number containing the case property value.

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
string myVal = myCase.GetPropertyNumber("PropertyName");
```

GetPropertyDateTime

This API will return the specified property of a Case object, if the property is a DateTime value. If the property is a string or a number, the API will return null.

Parameters

pPropName: The string name of the property to return.

Returns

object: A DateTime object containing the case property value.

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
string myVal = myCase.GetPropertyDateTime("PropertyName");
```

RecalcCaseInstanceName

This API will recalculate the name of the case instance to update any changes to the data used in the instance name, such as form field or case variables.

Parameters

None

Returns

None

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
myCase.RecalcCaseInstanceName();
```

SetPropertyText

This API will set the specified property of a Case object. If the case property is a numeric or DateTime value, Process Director will cast the string to the correct data type.

Parameters

pPropName: The string name of the property to return.

Returns

None

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
string myVal = "Value";
myCase.SetPropertyText("PropertyName") = myVal;
```

SetPropertyNumber

This API will set the specified property of a Case object, if the case property is a numeric value. Otherwise, the value will be set to null.

Parameters

pPropName: The string name of the property to return.

Returns

None

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
int myVal = 1;
myCase.SetPropertyNumber("PropertyName") = myVal;
```

SetPropertyDateTime

This API will set the specified property of a Case object, if the case property is a DateTime value. Otherwise, the value will be set to null.

Parameters

pPropName: The string name of the property to return.

Returns

None

Example

```
var myCase = Case.GetCaseByCASEINSTID(bp, "CASEINSTID");
datetime myDate = new DateTime(2023, 1, 18);
myCase.SetPropertyDateTime("PropertyName") = myDate;
```

ConditionSet Class

This object represents a Condition Set used for a Process Director object.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
ID	String	The string ID of the ConditionSet.

In addition to Properties, Condition Sets use the SystemVariableContext object to specify the various context properties for the Condition Set. Please see the SystemVariableContext topic for more information. Additionally, the Condition struct object is required as a parameter for some methods of this class. The Condition struct is documented below.

Methods

Copy

This API will create a copy of a ConditionSet. This is an overloaded method with the following possible declarations:

`public ConditionSet Copy()`

`public ConditionSet Copy(IEnumerable<Condition> cConditions)`

`public ConditionSet Copy(ContentObject oContainer)`

`public ConditionSet Copy (ContentObject oContainer, IEnumerable<Condition> cConditions)`

Parameters

cConditions: A iEnumerable Condition struct object that defines the conditions included in the ConditionSet.

oContainer: A ContentList object.

Returns

Rule object: Will return null if the ConditionSet isn't found.

Example

See the general Code Sample below.

Evaluate

This API will call the evaluation of the ConditionSet, returning the appropriate data. This is an overloaded method with the following possible declarations:

```
public bool Evaluate(SystemVariableContextReadonly Context)
```

```
public bool Evaluate(SysVarClass.IContextReadonly Context)
```

Parameters

Context: A SystemVariableContext object containing the context settings for the Condition Set.

Returns

string: The result of the Condition Set's evaluation.

Example

See the general Code Sample below.

Update

This API will update a ConditionSet.

Parameters

cConditions: A iEnumerable Condition struct object that defines the conditions included in the ConditionSet.

Returns

Rule object: Will return null if the ConditionSet isn't found.

Example

See the general Code Sample below.

Condition Struct

The Condition Struct object is an IEnumerable object that stores a condition or set of conditions.

Properties

These properties can be enumerated via a Get, but are Set privately, and can't be Set in script.

PROPERTY NAME	DESCRIPTION
Left	A SystemVariable object that defines the Left side of the condition.
Right	A SystemVariable object that defines the Right side of the condition.
Type	The type of comparison to make between the left and right side of the Condition.

Methods

Copy

This API will create a copy of a Condition.

Parameters

None

Returns

Condition object: A Clone of the specified Condition object.

Example

See the general Code Sample below.

Code Sample

A user can't instantiate a ConditionSet (or a Condition) object. Users will only interact with these objects through the "ReturnValues" property of a Business Rule. The following code evaluates a Business Rule, and then manually evaluates its return values, returning the first match:

```
var context = new SystemVariableContext {Form = BaseForm};
var sv = new SystemVariable(bp.eSysVar.FormField,
    new DataItem(FormEnums.eDataType.String),
    {String = "UserPicker1"});
sv.Parameters["format"] = "uid";
var sEval = sv.Evaluate(context);

var rule = Rule.GetRuleByRULEID(bp, "RULEID");
var resultRule = rule.Evaluate(context);

ReturnFirst(rule, context);

// Return the First matching value in a Rule
static string ReturnFirst(Rule rule, SystemVariableContext
```

```
context)
{
    foreach(var rv in rule.ReturnValues)
    {
        // The Copy below function is unnecessary, and added
only for illustration.
        // In actual use, the line below would simply be:
        // var cs = rv.Value();
        var cs = rv.Value.Copy();
        if (cs.Evaluate(context))
         return rv.Key.Evaluate(context);
    }
    return "";
}
```

ContentObject Class

This object represents the base class for all content objects (documents, Forms, Process Timelines, etc) stored in Process Director. All properties, methods, and events of this base class are available to every content object derived from this class.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
ContentPath	String	The full path including folders for this object
CreateTime	DateTime	The date/time the object was created
CreateUID	String	The UID of the user that created the object
Description	String	The description of this object
GroupName	String	The optional group name that this object is stored in (e.g. for Form attachments)
Icon	String	The Icon number of this object
ID	String	The ID of this object
Name	String	The name of this object
Parent_ID	String	The optional ID of the parent of this

PROPERTY NAME	DATA TYPE	DESCRIPTION
		object
PID	String	The internal Partition ID
Size	Integer	The size of this object, in Kilobytes.
Type	String	The description of this object
UpdateTime	DateTime	The date/time the object was last updated
UpdateUID	String	The UID of the user that last updated the object
Version	Integer	The version of the object. The version number is incremented for documents every time a document is checked in. For forms, the version number is incremented each time a user saves changes on the form.

PermObject Class

Since each object in the ContentObject Class has permissions that can be applied to the object, the PermObject class is used by the various permissions methods, such as SetPermissions, or GetPermissionsto list, get, set, and replicate permissions.

Properties

PROPERTY NAME	DESCRIPTION
PermID	The Permissions ID GUID string for the permission.
OID	The Object ID GUID string for the object to which the permission is applied.
GrantID	The Grant ID GUID string for the permissions grant. This is the user or group ID for the user or group to whom the grant is given.
PID	The Partition ID GUID string for the partition in which the object resides.
GrantType	The Grant Type of the permission. This is an object value

PROPERTY NAME	DESCRIPTION
	that enables you to set the object type to User or Group.
fRead	A boolean value to determine whether or not to grant read permission to the Content Object, i.e., an Object Definition.
fWrite	A boolean value to determine whether or not to grant write permission to the Content Object, i.e., an Object Definition.
fDelete	A boolean value to determine whether or not to grant delete permission to the Content Object, i.e., an Object Definition.
fExecute	A boolean value to determine whether or not to grant execute permission to the Content Object, i.e., an Object Definition.
fRead2	A boolean value to determine whether or not to grant read permission to the child objects of a Content Object, i.e., an Object Instance.
fWrite2	A boolean value to determine whether or not to grant write permission to the child objects of a Content Object, i.e., an Object Instance.
fDelete2	A boolean value to determine whether or not to grant delete permission to the child objects of a Content Object, i.e., an Object Instance.

> 🛑 **The values below are used only for Deny permissions. This won't be applicable to most installations. We strongly recommend that, due to system overhead, Deny permissions not be used unless necessary.**

fReadEx	A boolean value to determine whether or not to deny read permission to the Content Object, i.e., an Object Definition.
fWriteEx	A boolean value to determine whether or not to deny

PROPERTY NAME	DESCRIPTION
	write permission to the Content Object, i.e., an Object Definition.
fDeleteEx	A boolean value to determine whether or not to deny delete permission to the Content Object, i.e., an Object Definition.
fExecuteEx	A boolean value to determine whether or not to deny execute permission to the Content Object, i.e., an Object Definition.
fReadEx2	A boolean value to determine whether or not to deny read permission to the child objects of a Content Object, i.e., an Object Instance.
fWriteEx2	A boolean value to determine whether or not to denywrite permission to the child objects of a Content Object, i.e., an Object Instance.
fDeleteEx2	A boolean value to determine whether or not to deny delete permission to the child objects of a Content Object, i.e., an Object Instance.

Methods

Add

This API will add an object as child to the current object. This is an overloaded method, with the following possible method declarations:

```
public virtual bool Add(string pID)

public virtual bool Add(string pID, string pGroup)

public virtual bool Add(ContentObject pObj)

public virtual bool Add(ContentObject pObj, string pGroup)
```

Parameters

ContentObject: The ContentObject to add to this object.

pGroup: Optionally the group name to add into.

pID: The ID of the object to add.

Returns

boolean: True if the operation succeeds.

Example

```
CurrentForm.Add(Form.Instantiate(bp, "ObjectID"));
```

AddDocumentFromFS

This API will create a new document under an object from the local file system. This is an overloaded method with the following possible declarations:

```
public Document AddDocumentFromFS(string Path)
```

```
public Document AddDocumentFromFS(string Path, string Name)
```

```
public Document AddDocumentFromFS(Stream Stream, string Name)
```

```
public  Document  AddDocumentFromFS(Stream  Stream,  string  Name,
                                   string DocName)
```

Parameters

Path: The folder or full path to the local file.

Name: Optionally the file name in the path.

Stream: A FileSystem Stream object to add as a document

Returns

ContentObject: The new content object or null if the add fails.

Example

```
var oObject = Project.GetProjectbyPRID(bp, "PRID");
oObject.AddDocumentFromFS("c:\\documents\\doc1.doc");
```

AddDocumentFromBytes

This API will create a new document under an object from a byte array. This is an overloaded method with the following possible declarations:

```
public     Document     AddDocumentFromBytes  (string     Name,
                                   byte[] Bytes)
```

```
public     Document     AddDocumentFromBytes  (string     Name,
                                   byte     []              Bytes,
                                   bool Replace)
```

```
public      Document      AddDocumentFromBytes   (string        Name,
                                    byte    []              Bytes,
                                    bool                 Replace,
                                    bool SkipAutoWfStart)
```

Parameters

Name: Name of the new document.

Bytes: The array of bytes used to set the document contents.

Replace: Boolean value to direct if the document should replace another

SkipAutoWfStart: Boolean to determine whether to automatically start a process

Returns

ContentObject: The new content object or null if the add fails.

Example

```
byte[] DocumentData;
var oObject = Project.GetProjectbyPRID(bp, "PRID");
// Set document data from reading from file, web service call,
etc
// DocumentData = ...
oObject.AddDocumentFromBytes("My Doc.doc", DocumentData);
```

AddObjectMap

This API will add a shortcut to the object into the destination. This is an overloaded method with the following possible declarations:

```
public bool AddObjectMap(string DestinationID, ObjectType Destin-
ationObjectType)
```

```
public bool AddObjectMap(string DestinationID)
```

```
public bool AddObjectMap(ContentObject DestinationObj)
```

Parameters

ContentObject: The Content object to add a shortcut into.

DestinationID: The ID of the Content Object to add a shortcut into.

DestinationObjectType: The Type of the Content Object to add a shortcut into.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Document.GetDocumentbyDID(bp, "DID");
oObject.AddObjectMap("1234", ObjectType.Folder);
```

AddPending

This API will Add an object to the "Pending" queue, meaning that it will attach the object to the first Process which starts on it.

 This is an overloaded method with the following possible declarations:

```
public bool AddPending(ContentObject pObj)
```

```
public bool AddPending(ContentObject pObj, string pGroup)
```

Parameters

pObj: The Content object to add to the pending queue.

pGroup: The Group Name of the Object.

Returns

boolean: True if the operation succeeds.

Example

```
// Set the Process and content object
var oProcess = Project.GetProcessByID(bp, "PRID");
var oObject = Form.GetFormByFORMID("FORMID");
//Add the object to the Pending Queue
oProcess.AddPending(oObject);
```

AssignCategory

This API will set the Meta Data category for an object. This is an overloaded method with the following possible declarations:

```
public bool AssignCategory(string Category)
```

```
public static bool AssignCategory(bp BP, string ID, string Category)
```

Parameters

BP: The BP Logix environment class.

ID: The string ID of the Content Object.

Category: The string name of the category to which the object should be assigned.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.AssignCategory("CategoryName");
```

ConvertSysVarsInString

This API will convert system variables in a given string. This is an overloaded method with the following possible declarations:

```
public virtual string ConvertSysVarsInString(string pString)
```

```
public virtual string ConvertSysVarsInString (string pString,
                      SysVarClass.Context pContext)
```

```
public virtual string ConvertSysVarsInString (string pString,
                      SysVarClass.IContextReadonly pContext)
```

Parameters

pString: The string containing optional SysVars to convert.

pContext: A System Variable Context object identifying the context of the variables

Returns

string: The converted string

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
var NewString = oObject.ConvertSysVarsInString("Convert
{EMBEDDED_SYSVAR}");
```

CopyObject

This API will copy the Content Object to a destination. This is an overloaded method with the following possible declarations:

```
public      string      CopyObject   (string      DestinationID,
                      ObjectType DestinationObjectType)
```

```
public string CopyObject(string DestinationID)
```

```
public string CopyObject(string DestinationID, string GroupName)
```

Parameters

DestinationID: The ID of the Content Object to copy into.

DestinationObjectType: The Type of the Content Object to copy into. (optional parameter)

GroupName: The Group name of the object

Returns

string: The ID of the new object, or "" if the copy failed.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.CopyObject("DestinationID", "GroupName");
```

DeleteObject

This API will delete the Content Object.

Parameters

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.DeleteObject();
```

DeleteObjectAndChildren

This API call will delete a Content List instance object and all of its child objects. For instance, you can delete a form of process instance as well as all document attachments for those instances.

Parameters

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.DeleteObjectAndChildren();
```

GetAttribute

This API will get a specific attribute from the object's meta data. This is an over-loaded method with the following possible declarations:

```
public DataItem GetAttribute(string Category, string Attribute)
```

```
public static DataItem GetAttribute(bp BP, string ID, string Cat-
egory,
                                    string Attribute)
```

```
public DataItem GetAttribute(string pATTID)
```

```
public  static  DataItem  GetAttribute(bp  BP,  string  ID,  string
ATTID)
```

```
public DataItem GetAttribute(MetaCategory pCat, string Attribute)
```

```
public  static  DataItem  GetAttribute(bp  BP,  string  ID,  MetaCat-
egory                                                         Cat,
                                    string Attribute)
```

Parameters

Category: The Category of the Attribute to retrieve

Attribute: The Attribute name to retrieve

BP: The Process Director BP Object

ID: The ID of the attribute to retrieve

pCat: The Meta Data Category object

Cat: The Meta Data Category object of the Attribute re retrieve

ATTID: The Attribute Id of the Attribute to retrieve

Returns

DataItem: The data corresponding to the Attribute

Example

```
var oObject = Project.GetProjectbyPRID(bp, "PRID");
var dataAtt = oObject.GetAttribute("Category 1.Category 2",
    "Attribute Name");
var val = dataAtt.String;
```

GetAttributes

This API will get a list of attributes from the object's meta data. This is an over-loaded method with the following possible declarations:

```
public List<NameValueEx> GetAttributes()
```

```
public static List<NameValueEx> GetAttributes(bp BP, string ID)
```

```
public List<NameValueEx> GetAttributes(MetaCategory pCat)
```

```
public static List<NameValueEx> GetAttributes(bp BP, string ID,
                                    MetaCategory pCat)
```

```
public List<NameValueEx> GetAttributes(string pCat)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the attribute to retrieve

pCat: The Meta Data Category object

Returns

list: The Name-Value pairs of each Attribute

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
var atts = oObject.GetAttributes("Category 1");
foreach(var att in atts)
{
    var name = att.Name;
    var val = att.Value;
}
```

GetChildren

This API will create a list of all the children for the content object. This is an over-loaded method with the following possible declarations:

Gets All Children:

```
public virtual List<ContentObject> GetChildren()
public static List<ContentObject> GetChildren(bp BP, string Par-
entID)
```

Gets children of specified eMapType:

```
public    virtual    List<ContentObject>    GetChildren (bp.eMapType
MapType)
public     static     List<ContentObject>     GetChildren (bp     BP,
                                               string     ParentID,
                                               bp.eMapType
MapType)
```

Gets children of specified ObjectType:

```
public    virtual    List<ContentObject>    GetChildren (ObjectType
ObjectType)
public static List<ContentObject> GetChildren(bp BP, string Par-
entID,
                                                      ObjectType
ObjectType)
```

Gets children with a specified groupname:

```
public virtual List<ContentObject> GetChildren(string GroupName)
public static List<ContentObject> GetChildren(bp BP, string Par-
entID,
                                           string GroupName)
public    virtual    List<ContentObject>    GetChildren (ObjectType
ObjectType,
                                           string GroupName)
```

Gets children of specified ObjectType and eMapType:

```
public    virtual    List<ContentObject>    GetChildren (ObjectType
ObjectType,
                                           bp.eMapType
```

```
MapType,
                                    string GroupName)
public static List<ContentObject> GetChildren(bp BP, string Par-
entID,
                                    ObjectType
ObjectType,
                                    bp.eMapType
MapType,
                                    string GroupName)
```

Parameters

ObjectType: The optional type of objects to filter.

MapType: The optional map type of the objects to filter.

GroupName: The optional group name of the objects to filter.

ParentID: The optional ID of the parent object to filter

Returns

list<ContentObject>: A List of the ContentObject type.

Example

```
var childList = CurrentProject.GetChildren
(ObjectType.Document);
```

GetFileType

This API will return the file type for documents and Forms.

Parameters

Returns

string: The file extension (e.g. "docx", "ascx") of the document or Form.

Example

```
var oObject = Document.GetDocumentbyDID("DID");
var FileType = oObject.GetFileType();
```

GetObjectByID (Static Method)

This API will get a content object from the specified ID.

Parameters

BP: The Process Director BP Object

ID: The ID of the object to retrieve

Returns

ContentObject: Will return null if object isn't found.

Example

```
// Normally not used directly
var oObject = ContentObject.GetObjectByID(bp, "OID");
```

GetObjectByPathName (Static Method)

This API will get an object by its path.

Parameters

BP: The Process Director BP Object.

PartitionID: The Partition ID or name.

PathName: The complete path to the object to return.

Returns

ContentObject: The actual object or null if not found.

Example

```
// Get the Process Timeline named "My Timeline" in the folder
named "My Project"
// located in the "Partition1" partition.
var myTimeline = ContentObject.GetObjectByPathName(bp, "Par-
tition1",
    "/My Project/My Timeline");
```

GetPermissions

This API will return the permissions of the object. This is an overloaded method with the following possible declarations:

```
public static List<PermObject> GetPermissions(bp BP, string ID)
```

```
public List<PermObject> GetPermissions()
```

Parameters

BP: The Process Director BP Object.

ID: The ID of the object from which to retrieve permissions.

Returns

List<PermObject>: The list of permission records for the selected object.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
var PermList = oObject.GetPermissions();
```

GetProcess

This API will return the the process associated with a content object. This method is useful when you know the ID of a content object, such as a Form, but don't know the process with which the form is associated. This method will return either a Workflow or Project (Process Timeline) object, or null if the string ID of the content object isn't found. This is an overloaded method with the following possible declarations:

```
public Process GetProcess()
```

```
public Process GetProcess(string idProc)
```

Parameters

idProc: The string ID of the content object for which you wish to return the associated process.

Returns

Workflow or Project (Process Timeline) object: The process object associated with the idProc used as the input parameter.

Example

If a content object is ised in a single process, you can return the process without passing any parameters. In the sample below, a form is identified, then used to return the process associated with the form:

```
var form = Form.GetFormByFORMINSTID(bp, "FORMINSTID");
var proc = form.GetProcess();
```

If your content object is used in multiple processes, you'll need to pass the ProcessID as a parameter:

```
var form = Form.GetFormByFORMINSTID(bp, "FORMINSTID");
var proc = form.GetProcess("PRID");
```

IsCatAssigned

This API checks to see if an object has been assigned to a given Meta Data category. This is an overloaded method with the following possible declarations:

`public bool IsCatAssigned(string Category)`

`public static bool IsCatAssigned(bp BP, string ID, string Category)`

Parameters

BP: The Process Director BP Object.

ID: The ID of the ContentObject to check.

Category: The string name of the category to check.

Returns

boolean: True if the object is assigned to the given Meta Data Category.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.IsCatAssigned("CategoryName");
```

MoveObject

This API will move the Content Object to a new destination. This is an overloaded method with the following possible declarations:

`public bool MoveObject(string DestinationID, ObjectType DestinationObjectType)`

`public bool MoveObject(string DestinationID)`

`public bool MoveObject(ContentObject oDest)`

Parameters

DestinationID: The ID of the Content Object to move into.

DestinationObjectType: The Type of the Content Object to move into. (optional parameter)

oDest: The Content Object of the destination to which to move the object

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.MoveObject("1234");
```

RemoveCategory

This API removes an object from a given Meta Data category. This is an overloaded method with the following possible declarations:

```
public bool RemoveCategory(string Category)
```

```
public static bool RemoveCategory(bp BP, string ID, string Category)
```

Parameters

BP: The Process Director BP Object.

ID: The ID of the ContentObject to check.

Category: The string name of the category to check.

Returns

boolean: True if the method succeeds.

Example

```
var oObject = Form.GetFormByFORMID("FORMID");
oObject.RemoveCategory("CategoryName");
```

RemoveObjectFromParent

This API will remove the Content Object from its parent. This is an overloaded method with the following possible declarations:

```
public bool RemoveObjectFromParent()[Deprecated]
```

```
public bool RemoveObjectFromParent(ContentObject pParent)
```

```
public bool RemoveObjectFromParent (ContentObject pParent, bp.eMapType pMapType)
```

```
public bool RemoveObjectFromParent(string pParentID)
```

```
public bool RemoveObjectFromParent(string pParentID, bp.eMapType pMapType)
```

Parameters

pParent: The parent ContentObject.

eMapType : The relationship Map Type between the child and parent objects.

pParentID: The string ID of the parent object.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMINSTID(bp, "FORMINSTID");
oObject.RemoveObjectFromParent(pParentObject);
```

RemovePermissions

This API will remove all permissions for the object. Typically you'll then add permissions with the SetPermissions API. This is an overloaded method with the following possible declarations:

```
public bool RemovePermissions()
```

```
public static bool RemovePermissions(bp BP, string ID)
```

```
public bool RemovePermissions(Group group)
```

```
public bool RemovePermissions(User user)
```

```
public bool RemovePermissions(string ID)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the object from which to remove permissions.

Group: The Group object from which to remove permissions.

User: The user object from which to remove permissions.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.RemovePermissions(); // Remove all permissions to this
object
```

ReplicatePermsToChildren

This API will replicate all permissions for the object to all children of the object. This is an overloaded method with the following possible declarations:

```
public bool ReplicatePermsToChildren()
```

```
public static bool ReplicatePermsToChildren(bp BP, string ID)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the object from which to remove permissions.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.ReplicatePermsToChildren();
```

ReplicatePermsToChildrenAndForms

This API will replicate all permissions for the object to all children of the object. Additionally, if any of the children are form instances, the permissions will be replicated to their children as well. This is an overloaded method with the following possible declarations:

```
public bool ReplicatePermsToChildrenAndForms()
```

```
public static bool ReplicatePermsToChildrenAndForms(bp BP, string ID)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the object from which to remove permissions.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.ReplicatePermsToChildrenAndForms();
```

SetAttribute

This API will set a specific attribute in the object's Meta Data. This is an overloaded method with the following possible declarations:

public bool SetAttribute (string Category, string Attribute, string Value)

public bool SetAttribute (string Category, string Attribute, DataItem Value)

public static bool SetAttribute (bp BP, string ID, string Category,
                                 string   Attribute,
                                 string Value)

public static bool SetAttribute (bp BP, string ID, string Category,
                                 string   Attribute,
                                 DataItem Value)

public static bool SetAttribute(bp BP, string ID, string ATTID, string Value)

public static bool SetAttribute(bp BP, string ID, string ATTID, DataItem Value)

Parameters

BP: The Process Director BP Object

ID: The ID of the object from which to remove permissions.

Category: The Category of the Attribute to set.

Attribute: The Attribute name to set.

Value: The string or DataItem to assign to the Attribute.

ATTID: The ID of the attribute from which to remove permissions.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.SetAttribute("Category 1.Category 2", "Attribute
```

```
Name", "1");
```

SetAttributes

This API will set a list of attributes in the object's Meta Data. This is an overloaded method with the following possible declarations:

```
public bool SetAttributes(List<NameValue> Attributes)

public   static   bool   SetAttributes (bp   BP,   string   ID,
                                List<NameValue> Attributes)

public bool SetAttributes(List<NameValueEx> Attributes)

public   static   bool   SetAttributes (bp   BP,   string   ID,
                                List<NameValueEx> Attributes)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the object from which to remove permissions.

Attributes: A list of NameValueEx objects where the Name specifies the `Cat-egory.Attribute` name, and the Value specifies the value to set.

Returns

boolean: True if the operation succeeds.

Example

```
MetaCategory myCat = GetCategory(bp, "PID", "Category 1");
var oObject = Form.GetFormByFORMID(bp, "FORMID");
var atts = myCat.GetAttributes("Category 1.Category 1A");
foreach(var att in atts)
{
    var name = att.Name;
    var val = att.Value;
}
oObject.SetAttributes(atts);
```

SetExternalAttribute

This API will set a Meta Data attribute for an object based on the external name. This is an overloaded method with the following possible declarations:

```
 public   bool   SetExternalAttribute (string   ExternalName,   string
Value)
```

```
public   static   bool   SetExternalAttribute (bp  BP,   string   ID,
                                       string       ExternalName,
                                       string Value)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the object to which to set the attribute.

ExternalName: The string value for the external name to use for the attribute name.

Value: The string value of the Attribute to set.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.SetExternalAttribute("ExternalName", "ValueToSet");
```

SetExternalAttributes

This API will set Meta Data attributes for an object based on a list of external names. This is an overloaded method with the following possible declarations:

```
public bool SetExternalAttributes(List<NameValue> MetaData)
```

```
public   static   bool   SetExternalAttributes (bp  BP,   string   ID,
                                       List<NameValue>
MetaData)
```

Parameters

BP: The Process Director BP Object

ID: The ID of the object to which to set the attributes.

MetaData: A list object consisting of Name/Value pairs.

Returns

boolean: True if the operation succeeds.

Example

```
List<NameValue> atts = new List<NameValue>();
atts.Add("Attr1");
atts.Add("Attr2");
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.SetExternalAttributes(atts);
```

SetGroupName

This API will set the group that the object is in.

Parameters

GroupName: The group for this object.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.SetGroupName("My Group");
```

SetMetaData

This API will set the Meta Data categories and attributes for an object using an XML string. This is an overloaded method with the following possible declarations:

`public bool SetMetaData(string MetaData)`

`public static bool SetMetaData(bp BP, string ID, string XML_Data)`

Parameters

MetaData: The XML String containing the Meta Data information.

BP: The BP Logix environment.

ID: The ID of the object for which the Meta Data is to be set..

XML_Data: The XML String containing the Meta Data information.

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");

//Make the XML String for the Meta Data
StringBuilder sb = new StringBuilder();
sb.Append("<?xml version=\"1.0\" encoding=\"utf-8\"?>");
sb.Append("<META_DATA xmlns:xsi=");
sb.Append("\"http://www.w3.org/2001/XMLSchema-instance\"");
sb.Append
("xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">\"");
sb.Append("<INPUT_META_DATA>");
sb.Append("<ATTRIB_NAME>Cat 1.Cat 2.Attrib 1</ATTRIB_NAME>");
sb.Append("<META_VALUE>Value 1</META_VALUE>");
sb.Append("</INPUT_META_DATA>");
sb.Append("<INPUT_META_DATA>");
sb.Append("<ATTRIB_NAME>Cat 1.Cat 2.Attrib 2</ATTRIB_NAME>");
sb.Append("<META_VALUE>Value 2</META_VALUE>");
sb.Append("</INPUT_META_DATA>");
sb.Append("</META_DATA>");

//Set Meta Data
oObject.SetMetaData(sb.ToString());
```

SetPermissions

This API will set permissions available on the object for a specific User or Group of Users. This is an overloaded method with the following possible declarations:

```
public bool SetPermissions(User GrantUser, bool PermView, bool PermModify,
                           bool PermDelete, bool PermRun)
public bool SetPermissions(Group GrantGroup, bool PermView, bool PermModify,
                           bool PermDelete, bool PermRun)
public bool SetPermissions(string GrantID, ObjectType GrantType,
                           bool PermView, bool PermModify,
                           bool PermDelete, bool PermRun)
public bool SetPermissions(string Partition, string GrantID,
                           ObjectType GrantType, bool PermView,
                           bool PermModify, bool PermDelete, bool
PermRun)
```

```
public static bool SetPermissions(bp BP, string ID, string PID,
                                  string              GrantID,
                                  ObjectType          GrantType,
                                  bool                PermView,
                                  bool                PermModify,
                                  bool PermDelete, bool PermRun)
```

Parameters

User/Group: The User or Group for which to set the permissions on the object.

PermView: Whether or not the user will have View permission on the object.

PermModify: Whether or not the user will have Modify permission on the object.

PermDelete: Whether or not the user will have Delete permission on the object.

PermRun: Whether or not the user will have Run permission on the object.

BP: The Process Director BP Object

ID: The string ID of the Process Director object whose permissions should be set.

PID: The partition ID of the partition where the object resides.

GrantID: The string ID of the user/group for whom permissions should be added.

GrantType: The GrantType object of the permissions.

Returns

boolean: True if the operation succeeds.

Example

```
var user = User.GetUserByUserID(bp, "User 1");
var oObject = Form.GetFormByFORMID(bp, "FORMID");
// Grant "User 1" permission to view, modify, or run the
object
// (but not to delete it)
oObject.SetPermissions(user, true, true, false, true);

// Grant the "Finance" group view permissions only
var group = Group.GetGroupByName(bp, "Finance");
oObject.SetPermissions(group, true, false, false, false);
```

SetPermissionsEx

This API will set permissions available on the object for a specific User or Group of Users.

```
public    static    bool    SetPermissionsEx (bp    BP,    string    ID,
                                            string                PID,
                                            string              GrantID,
                    ObjectType                              GrantType,
                                            short               PermView,
                                            short             PermModify,
                                            short             PermDelete,
                                            short PermRun)
```

Parameters

BP: The Process Director BP Object

ID: The string ID of the Process Director object whose permissions should be set.

PID: The partition ID of the partition where the object resides.

GrantID: The string ID of the user/group for whom permissions should be added.

GrantType: The Value of the Perm.GrantType object.

PermView: Whether or not the user will have View permission on the object.

PermModify: Whether or not the user will have Modify permission on the object.

PermDelete: Whether or not the user will have Delete permission on the object.

PermRun: Whether or not the user will have Run permission on the object.

Returns

boolean: True if the operation succeeds.

Example

```
var granttype= TempPerm.GrantType;
var oObject = Form.GetFormByFORMID(bp, "FORMID");
// Grant "User 1" permission to view, modify, or run the
object
// (but not to delete it)
oObject.SetPermissionsEx(bp, "IDValue", "PartitionID",
"GrantID",
    granttype, 1, 1, 0, 1);
```

UpdateObject

This API will update the name and description properties in the database

Parameters

Returns

boolean: True if the operation succeeds.

Example

```
var oObject = Form.GetFormByFORMID(bp, "FORMID");
oObject.UpdateObject();
```

DataSource Class

This object represents a database DataSource which contains the connection details to connect to a database. The Provider and Connection can be used with native ADO.NET calls to connect to external databases.

This object is derived from the ContentObject class. All properties and methods from the ContentObject are supported for this object, plus the properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Provider	String	The Provider string of this connection.
Connection	String	The connection string for this SQL connection.
DBType	String	The optional type of data-base.

Methods

CacheCount (Static Method)

This API will return the number of items in the Datasource cache.

Parameters

DSID: The string ID of the DataSource.

Returns

Integer: The Number of items in the cache.

Example

```
int count = DataSource.CacheCount("DSID");
```

ClearCache (Static Method)

This API will remove all items from the Datasource cache.

Parameters

DSID: The string ID of the DataSource.

Returns

Boolean: Returns "true" if the operation succeeds.

Example

```
bool cleared = DataSource.ClearCache("DSID");
```

GetDataSourceByDSID (Static Method)

This API will get a DataSource object from the specified ID.

Parameters

BP: The Process Director BP Object

DSID: The ID of the DataSource to retrieve.

Returns

DataSource: Will return null if Datasource isn't found.

Example

```
// Normally not used directly
var oDS = DataSource.GetDataSourceByDSID(bp, "DSID");
```

GetDataSourceByName (Static Method)

This API will get a DataSource object from the specified ID.

Parameters

BP: The Process Director BP Object

Name: The name of the DataSource to retrieve.

Returns

DataSource: Will return null if Datasource isn't found.

Example

```
// Get the Datasource to my ERP system
var oDS = DataSource.GetDataSourceByName(bp, "ERP System");
```

DocumentObject Class

This object represents a Document.

This object is derived from the ContentObject class. All properties and methods from the ContentObject are supported for the Document object, plus the properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
FileType	String	The file extension of the document
DOS_FileName	String	The DOS "friendly" file name of the document
CheckedOut	Boolean	Is the document currently checked out?
CheckedOutUID	String	If this document is checked out, the UID of the user
FilePath	String	This property will return the physical file path of an attachment that is stored on the file system. This property will return no value if the document is stored in the Process Director database.
WebViewableOID	String	The OID of the optional web viewable object
ManagedOID	String	The OID of the actual document object

Methods

CreateThumbnail

This method implements a simple routine that creates a thumbnail image of the first page of a document.

Parameters

BP: The Process Director BP Object.

DID: The ID of the document to retrieve.

Returns

N/A

Example

```
BPLogix.WorkflowDirector.SDK.Document.CreateThumbnail(bp,
"DID");
```

GetBytes

This API will return the bytes for a document.

Parameters

None

Returns

byte: Will return null if document isn't found.

Example

```
byte[] DocBytes = Document.GetBytes();
```

GetDocumentByDID (Static Method)

This API will get a document object from the specified ID.

Parameters

BP: The Process Director BP Object.

DID: The ID of the document to retrieve.

Returns

Document: Will return null if document isn't found.

Example

```
// Normally not used directly
var oDocument = Document.GetDocumentByDID(bp, "DID");
```

GetStream

This API will return the stream for a document.

Parameters

None

Returns

stream: Will return null if document isn't found.

Example

```
stream DocBytes = Document.GetStream();
```

RemoveThumbnail

This method implements a simple routine that removes a thumbnail image.

Parameters

BP: The Process Director BP Object.

DID: The ID of the document to retrieve.

Returns

N/A

Example

```
BPLogix.WorkflowDirector.SDK.Document.RemoveThumbnail(bp,
"DID");
```

SetDocReviewable

This API will set the web viewable object for a document

Parameters

SourcePath: The full path to the local file system of the web viewable object.

Returns

Boolean: True if the operation succeeds.

Example

```
BPLogix.WorkflowDirector.SDK.Document.SetDocReviewable("c:\\Im-
port\\File.pdf");
```

UpdateDocData

This API enables you to set or replace the contents of a documen.

Parameters

pData: The Stream object to be updated.

Description (Optional): The String description of the new document data.

Returns

Boolean: True if the operation succeeds.

Example

This example overwrites a document in the Content List from a document stored in a local file system.

```
Document doc = Document.GetDocumentByDID(bp, "DID")
Stream pData = File.Open("C:\Filepath\file.txt",
FileMode.Open);
doc.UpdateDocData(pData);
```

WriteDocumentToDisk

This API will write the document to a local file.

Parameters

DestPath: The full path of the local output file.

Returns

Boolean: True if the operation succeeds.

Example

```
Document oDocument = Document.GetDocumentByDID(bp, "DID")
oDocument.WriteDocumentToDisk("c:\\Documents\\SavedDoc.docx");
```

Dropdown Object Class

This object is derived from the ContentObject class. This object represents a Content Object of the DropDown type. It contains a list of name/value pairs which can

fill a DropDown on a Form.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Values	List Object	The list of name/value pairs for the dropdown as a List<DropDownValue> class
DDID	String	The DropDown ID of the DropDown Content Object (same as ID property)

Methods

GetDropDownByDDID (Static Method)

This API will return a DropDownObject which corresponds to the ID you pass it.

Parameters

BP: The Process Director BP Object.

DDID: The ID of the Dropdown Object to retrieve

Returns

DropDownObject: An instance of the DropDownObject, or null if no DropDown could be found.

Example

```
var cDD = CurrentForm.FormControl("DropDownPick").Value;
// if "DropDownPick" is a ContentPicker on a Form
var dd = DropDownObject.GetDropDownByDDID(bp, cDD);
// Now we can use dd for our DropDownObject
bp.log0("Number of entries in the DropDown: " + dd.Val-
ues.Count);
```

SetDropDownValues

This API sets the list of DropDown name/value pairs.

Parameters

Values: The list of name/value pairs for the DropDown as a List<DropDownValue> (or other IEnumerable) class.

Returns

Boolean: Whether or not the call could set the DropDownObject's values.

Example

```
var ddvlist = new List<DropDownValue>();
ddvlist.Add(new DropDownValue("[Select a State]", ""));
ddvlist.Add(new DropDownValue("California", "CA"));
ddvlist.Add(new DropDownValue("Texas", "TX"));
var cDD = CurrentForm.FormControl("DropDownPick").Value;
// If "DropDownPick" is a ContentPicker on a Form
var dd = DropDownObject.GetDropDownByDDID(bp, cDD);
dd.SetDropDownValues(ddvlist);
// Will set DropDown to contain:
// "[Select a State]:''",
// "California:CA",
// "Texas:TX"
```

DropdownValue Object Class

This object represents a single name/value pair for use in a DropDown and similar controls.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Text	String	The text to display for the DropDown entry
Value	String	The actual value to use when using the DropDown entry

Methods

GetDropDownValues (Static Method)

This API will return the list of name/value pairs for a dropdown as a List<DropDownValue> class.

Parameters

BP: The Process Director BP Object

DDID: The ID of the DropDown Object from which to retrieve the DropDownValue List.

Returns

List<DropDownValue>: The list of name/value pairs for the DropDown.

Example

```
var cDD = CurrentForm.FormControl("DropDownPick").Value;
// If "DropDownPick" is a ContentPicker on a Form
var ddlist = DropDownValue.GetDropDownValues(bp, cDD);
bp.log0("Number of entries in the DropDown: " + ddlist.Count);
```

SetDropDownValues

This API sets the list of DropDown name/value pairs.

Parameters

Values: The list of name/value pairs for the DropDown as a List<DropDownValue> (or other IEnumerable) class.

Returns

Boolean: Whether or not the call could set the DropDownObject's values.

Example

```
var ddvlist = new List<DropDownValue>();
ddvlist.Add(new DropDownValue("[Select a State]", ""));
ddvlist.Add(new DropDownValue("California", "CA"));
ddvlist.Add(new DropDownValue("Texas", "TX"));
var cDD = CurrentForm.FormControl("DropDownPick").Value;
// if "DropDownPick" is a ContentPicker on a Form
var dd = DropDownObject.GetDropDownByDDID(bp, cDD);
dd.SetDropDownValues(ddvlist);
// Will set DropDown to contain
// "[Select a State]:''",
// "California:CA",
// "Texas:TX"
```

Constructor

Parameters

Text: The text to display for the DropDown entry

Value (Optional): The actual value to use when using the DropDown entry. If the constructor doesn't receive this parameter, it will use the 'Text' parameter for the Value as well.

Example

```
// Create a value with Text:"[Select a Value]" and Value:""
var ddv1 = new DropDownValue("[Select a Value]", "");

// Creates a value with Text:"Item1" and Value:"Item1"
var ddv2 = new DropDownValue("Item1");

// Creates the same DropDownValue as above ("Item1"/"Item1")
var ddv3 = new DropDownValue("Item1", "Item1");
```

Excel Class

This object represents an Excel document object.

Methods

GetValueByCell

Returns a cell value from an Excel Spreadsheet. This is an overloaded method with the following possible declarations:

public static string GetValueByCell (bp BP, string ExcelPath, string CellName)

public static string GetValueByCell (bp BP, Stream ExcelStream, string CellName)

Parameters

bp: The BP Logix Object.

ExcelPath: The file system path string to the Excel file.

ExcelStream: The stream object containing the stream of the Excel file.

CellName: The string name of the cell location.

Returns

List: A string value from the cell.

Example

```
string ExcelValue = Excel.GetValuesByCell(bp, "C:\\FilePath",
"R1C2");
```

GetValuesByCell

Returns a List object containing cell values for a given number of rows in an Excel spreadsheet. This is an overloaded method with the following possible declarations:

```
public   static   List<string>   GetValuesByCell (bp   BP,   string
ExcelPath,

                                        string  CellName,  int
pRows)
public   static   List<string>   GetValuesByCell (bp   BP,   Stream
ExcelStream,

                                        string  CellName,  int
pRows)
```

Parameters

bp: The BP Logix Object.

ExcelPath: The file system path string to the Excel file.

ExcelStream: The stream object containing the stream of the Excel file.

CellName: The string name of the cell location.

pRows: The integer number of rows to return.

Returns

List: A list object containing the string values from the cell rows.

Example

```
// Return the values from four rows in the excel file.
List ExcelValues = Excel.GetValuesByCell(bp, "C:\\FilePath",
"R1C2", 4);
```

GetValueByRangeName

Returns a List object containing cell values for a given number of rows in an Excel spreadsheet. This is an overloaded method with the following possible declarations:

```
public static string GetValueByRangeName(bp BP, string ExcelPath,
                                        string RangeName)
public   static   string   GetValueByRangeName (bp   BP,   Stream
ExcelStream,

                                        string RangeName)
```

Parameters

bp: The BP Logix Object.

ExcelPath: The file system path string to the Excel file.

ExcelStream: The stream object containing the stream of the Excel file.

RangeName: The named range from which to pull the value.

Returns

List: A list object containing the string values from the cell rows.

Example

```
string ExcelValue = Excel.GetValueByRangeName(bp,
"C:\\FilePath",
    "NamedRange");
```

GetValuesByRow

Returns a List object containing cell values for a given row in an Excel spreadsheet.

Parameters

bp: The BP Logix Object.

Stream: The stream object containing the stream of the Excel file.

pRow: The integer row number from which to pull the values.

Returns

List: A list object containing the string values from the row.

Example

```
string ExcelValue = Excel.GetValuesByRow(bp, [StreamObject],
[RowInteger]);
```

Folder Class

This object represents a Folder, and is derived from the ContentObject class. All properties and methods from the ContentObject are supported for this object, plus the properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
ID	String	The ID of this specific form instance
Name	String	The text name of this specific form instance
PID	String	The Partition ID of the partition where this form resides
CreateTime	DateTime	The date/time of this specific form instance's creation
UpdateTime	DateTime	The most recent date/time that a user changed this specific form instance
CreateUID	String	The User ID of the user which created this specific form instance
UpdateUID	String	The User ID of the user who most recently update this specific form instance

Methods

CreatePath (Static Method)

This API will create the complete folder path for the specific input.

Parameters

BP: The Process Director BP Object

PartitionID: The ID of the partition to use.

Path: The complete path to of the folder structure to create.

Returns

Folder: The newly created folder.

Example

```
// Normally not used directly
var oFolder = Folder.CreatePath(bp, CurrentPartition, "/folder-
1/folder2");
```

CreateNewFolder (Static Method)

This API will create a new folder in the specified parent.

Parameters

BP: The Process Director BP Object

ParentID: The ID of the parent.

FolderName: The name of the new folder to create.

Returns

Folder: The newly created folder.

Example

```
// Normally not used directly
var oFolder = Folder.CreateNewFolder(bp, SomeParentID, "New
Folder");
```

CreateSubFolder

This API will create a sub folder under the current folder

Parameters

FolderName: The name of the folder to create.

Returns

Folder: The newly created folder.

Example

```
var oFolder = Folder.CreateNewFolder(bp, SomeParentID, "New
Folder");
var NewFolder = oFolder.CreateSubFolder("Subfolder Name");
```

GetFolderByID (Static Method)

This API will get a Folder from the specified ID.

Parameters

BP: The Process Director BP Object.

FID: The ID of the folder to retrieve.

Returns

Folder: The actual folder or null if not found.

Example

```
Var oFolder = Folder.GetFolderByID (bp, "FolderID");
```

GetFolderByPathName (Static Method)

This API will get a Folder object from the specified Folder Path.

Parameters

BP: The Process Director BP Object

PID: The Partition ID or name.

PathName: The complete path of folder to retrieve.

Returns

Folder: Will return null if partition isn't found.

Example

```
var oFolder = Folder.GetFolderByPathName (bp,
CurrentPartition, "/folder1/folder2");
```

Form Class

This object represents a Form instance. An instance is a completed Form, or one that is currently being edited.

When developing Form scripts (in the various callback methods such as BP_Event) or Timeline scripts, you are automatically given an instance of the "current" Form with the CurrentForm variable.

This object is derived from the ContentObject class. All properties and methods from the ContentObject are supported for this object, plus the properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
CloseOptions	Code Enum	This can be set in any form script. This property can be set to the following: `FormCloseOptions.CancelAndClose` This will cancel the form when script returns.

PROPERTY NAME	DATA TYPE	DESCRIPTION
		`FormCloseOptions.SaveAndClose` This will save and close the form when script returns. `FormCloseOptions.NoneNo   Action` (default)
ControlFocus	Form Control Object	This property can be used to set the focus to a named control after the event returns. Set this property to null to prevent the focus from being set. `CurrentForm.ControlFocus = CurrentForm.FormControl("Text1");`
CreateTime	DateTime	The date/time of this specific form instance's creation
CreateUID	String	The User ID of the user which created this specific form instance
CustomOnLoad	JavaScript Function Name	Enables you to specify a block of custom JavaScript to run in the OnLoad event of the Form. For example, in an HTML Code control on the Form, you might specify the JavaScript to run, then set the CustomOnload property as follows in the Raw HTML property of the control: ```<script>\nfunction loadAlert ()\n{\n alert("Hello, World!")\n}\n\nCustomOnLoad = loadAlert();\n</script>```

PROPERTY NAME	DATA TYPE	DESCRIPTION
		ⓘ **As always, when using JavaScript, ensure the HTML Code control's name is blank, to prevent Process Director from trying to interpret the curly brackets as a system variable.**
		In this example, the code above will display the alert box when the form loads.
DOCTYPE_HTML	Boolean	If set to true will use HTML rather than XHTML
DOCTYPE_HTML5	Boolean	If set to true will use HTML rather than XHTML
FormCase	Case Instance Object	Returns the Case instance associated with a Form instance, e.g.: `var currCase = CurrentForm.FormCase;`
FORMID	String	The ID of the Form Definition for this instance
FORMINSTID	String	The optional ID of the Form instance
FormPartition	Partition Object	Set to the Partition object of the Forms partition
ID	String	The ID of this specific form instance
Name	String	The text name of this specific form instance
PID	String	The Partition ID of the partition where this form resides
ReturnNullsForErrors	Boolean	If set to true, APIs such as FormControl will return a null if the form control isn't found. Otherwise an empty class will be

PROPERTY NAME	DATA TYPE	DESCRIPTION
		returned. The default is false.
SkipSetFocus	Boolean	Skips the set focus event.
UpdateTime	DateTime	The most recent date/time that a user changed this specific form instance
UpdateUID	String	The User ID of the user who most recently update this specific form instance

Methods

AddErrorMessage

This API will add an error message to the form which will be displayed to the user. If an error message is added, the user won't be able to submit the form. This is an overloaded method with the following possible declarations:

```
public    void    AddErrorMessage (FormMessageString    pMessage,
                        params object[] pParams)
```

```
public  void  AddErrorMessage (string  pFormat, params  object []
pParams)
```

```
public T AddErrorMessage<T>(T ret, string pFormat, params object
[] pParams)
```

```
public  T  AddErrorMessage<T> (T  ret,  FormMessageString  pMessage,
                        params object[] pParams)
```

Parameters

pMessage: The error message to return (this can be a string or the FormMessageString class).

pFormat: The format string for the error message.

pParams: A parameters object containing the message parameters.

Returns

None

Example

```
// This example will be called in the validation event of a
form script
protected override void BP_Validation()
{
    // Display an error on the form if the Amount form field
is > 100
    // and place the focus on the form field "Amount"
    If (CurrentForm.FormControl("Amount").Number > 100)
    {
        CurrentForm.AddErrorMessage(new FormMessageString
("Must be < 100", "Amount"));
    }
}
```

AddInfoMessage

This API will add an informational message to the form which will be displayed to the user. If an informational message is added, the user won't be able to submit the form. This is an overloaded method with the following possible declarations:

public   void   AddInfoMessage (FormMessageString   pMessage,
                                  params object[] pParams)

public   void   AddInfoMessage (string   pFormat,   params   object []
pParams)

Parameters

pMessage: The error message to return (this can be a string or the FormMessageString class).

pParams: A parameters object containing the message parameters.

Returns

None

Example

```
// This example will be called in every event
protected override void BP_Event()
{
    // Display a message on the form if the Amount form field
is > 100
    If (CurrentForm.FormControl("Amount").Number > 100)
    {
        CurrentForm.AddInfoMessage("The amount is > 100");
    }
}
```

AddJavaScript

This API adds a block of JavaScript code to the form.

Parameters

JavaScript: The actual JavaScript code to add.

Returns

None

Examples

```
// We'll add our JavaScript in the initial load
protected override void BP_ViewStateiInit()
{
    CurrentForm.AddJavaScript("alert('Please submit the form
before
        closing it.')");
}
```

ClearJavaScript

This API clears all JavaScript code from the form.

Parameters

None.

Returns

None

Examples

```
// We'll clear our JavaScript in the initial load
protected override void BP_ViewStateiInit()
{
    CurrentForm.ClearJavaScript();
}
```

AddTopHTML

This API adds a block of HTML code to the top of a form.

Parameters

pHTML: A string value containing the raw HTML code to add to the top of the page.

Returns

None

Examples

```
// We'll add our JavaScript in the initial load
protected override void BP_ViewStateiInit()
{
    CurrentForm.AddTopHTML("<p>Hello, World!</p>");
}
```

AddBottomHTML

This API adds a block of HTML code to the bottom of a form.

Parameters

pHTML: A string value containing the raw HTML code to add to the top of the page.

Returns

None

Examples

```
// We'll add our JavaScript in the initial load
protected override void BP_ViewStateiInit()
{
    CurrentForm.AddBottomHTML("<p>Hello, World!</p>");
}
```

ConvertSysVarsInString

This API converts system variables in a string.

Parameters

String: The string in which to find the system variables.

FormControl: (Optional) The form control to reference for control-specific system variables (e.g. "{ROW_NUM}")

Returns

String: The resultant string after converting every system variable.

Examples

```
// Returns "USER1 editing this form on 2008-01-01"
bp.log0(CurrentForm.ConvertSysVarsInString("{CURR_USER} edit-
ing this form
 on {CURR_DATE}"));
var cText = CurrentForm.FormControl("Item", 2);

// Returns "Row number: 2"
string result2 = CurrentForm.ConvertSysVarsInString("Row num-
ber: {ROW_NUM}", cText);
```

CreateDummy

This API creates a dummy copy of an existing form, or of an Array Row, which can be filled with data attached to a Process Timeline via other APIs. This is an over-loaded method with the following possible declarations:

public static FormControl CreateDummy(Form form)

public static FormControl CreateDummy(FormControl fcArray, int row)

Parameters

Form: The Form object to reproduce.

fcArray: An Array control.

row: An Array row.

Returns

None

Examples

```
Form myForm = CreateDummy(CurrentForm);
```

DoValidation

This API initiates FormValidation.

Parameters

SkipValidationRules: Boolean to determine whether to skip the validation rules for the form.

OnlyCheckEventField: Boolean to determine whether to validate only the event field.

Returns

None

Example

```
CurrentForm.DoValidation(false, false);
```

FindControlInForms (Static Method)

This API searches through each field of a given name containing a given value and returns a list of each form control that matches the query. This is an overloaded method with the following possible declarations:

```
public static List<FormControl> FindControlInForms(bp BP, string
FCName,
                                                   string Value)
public static List<FormControl> FindControlInForms(bp BP, string
FCName,
                                                   string FORMID,
                                                   string Value)
```

Parameters

BP: The BP Object.

FCName: The name of the form controls that will be returned.

FORMID (Optional): The FORMID of the form definition that contains the controls.

Value: The value of the form controls that will be returned.

Returns

List: A list object containing the form controls.

Example

```
List myList = FindControlInForms(bp, "ControlName", "Con-
trolValue");
```

FormControl

This API will return a single form control. When using arrays, if you retrieve a form object without specifying a row number, you are acting on the entire column. This is an overloaded method with the following possible declarations:

```
public FormControl FormControl(string Name, int ArrayNum)
```

```
public FormControl FormControl(string pName, FormControl RefCon-
trol)
```

```
public   FormControl   FormControl (string   pName,   FormControl
RefArray,
                              int ArrayNum)
```

Parameters

Name: The name of the form control to retrieve.

ArrayNum: (optional) Form controls in an array, the row number of the specific control

ReturnNullIfNotFound: (optional) If set to true, the API will return a null if the form control isn't found. If set to false (default), the API will return an empty FormControl object if it isn't found

Returns

FormControl: The form control instance.

Example

```
// Add 1 to the Count form control
CurrentForm.FormControl("Amount").Number += 1;

// Make the Item in row 2 required
CurrentForm.FormControl("Item", 2).Display.Required =
eYNU.Yes;

// Make the entire Description column required
CurrentForm.FormControl("CodeComment").Display.Required =
eYNU.Yes;
```

FormControls

This API will return the list of all form controls for this form instance.

Parameters

Message: The actual error message to add

Returns

List<FormControl> The list of all form controls for this form instance.

Example

```
var formControls = BaseCurrentForm.FormControls;
```

FormControlByID

This API will return a form control which corresponds to the ID you pass it. You would use this when you have a Form's ID (e.g. from a ControlPicker control). This is an overloaded method with the following possible declarations:

public FormControl FormControlByID(string FCID)

public FormControl FormControlByID(string FCID, int ArrayNum)

public FormControl FormControlByID (string pFCID, FormControl RefControl)

Parameters

ArrayNum: (optional) Form controls in an array, the row number of the specific control

FCID: The ID of the control to retrieve

pFCID: The ID of the control to retrieve

RefControl: Optional) The FormControl object to be retrieved

Returns

FormControl: The form control instance.

Examples

```
var cPicker = CurrentForm.FormControl("CodeComment").Value;
// If "ControlsPick" is a ControlPicker on the Form
// set the form control's text to "Add Another"
CurrentForm.FormControlByID(cPicker).Text = "Add Another";

var cPick2 = CurrentForm.FormControl("CodeComment2").Value;
// Increment the control's value by 1
CurrentForm.FormControlByID(cPick2).Number += 1;
// Disable the whole column of the control
CurrentForm.FormControlByID(cPick2).Display.Enabled = false;
```

FormNavigate

This API is an **override** for the base FormNavigate method, and enables you to create a custom Form Navigate method.

```
public virtual void FormNavigate(BPLogix.WorkflowDirector.SDK.bp
bp, string URL)
```

Parameters

bp: The Process Director SDK handle.

URL: The string value containing the URL to which to navigate.

Returns

None.

Examples

```
public virtual void FormNavigate(BPLo-
gix.WorkflowDirector.SDK.bp bp, string URL)
{
    // Your custom page navigation code here.
}
```

GetErrorMessages

This API will get the list of all current error messages. The error messages may have been placed there from the built-in validation, or from messages added from

script.

Parameters

None

Returns

List<FormMessageString>: The list of form error messages.

Example

```
// This example will be called after built-in validation has
occurred
protected override void BP_Validation_Post()
{
    var ErrorList = CurrentForm.GetErrorMessages();
    // Check to see if there are more than 5 error messages
    if (ErrorList.Count > 5)
    {
        CurrentForm.AddInfoMessage("More than 5 errors!");
    }
}
```

GetFormByFORMID (Static Method)

This API will get a Form object based on the Form ID.

Parameters

BP: The BP environment.

FORMID: The ID of the form to retrieve.

Returns

Form: An instance of the form object, or null if it couldn't be found.

Example

```
// This example will get a form instance, and log its name
var OtherForm = Form.GetFormByFORMID(bp, "FORMID");
bp.log0("The form name is: " + OtherForm.Name);
```

GetFormByFORMINSTID (Static Method)

This API will get a Form instance object based on the ID.

Parameters

BP: The BP environment.

FORMINSTID: The ID of the form instance to retrieve

Returns

Form: An instance of the form object, or null if it couldn't be found.

Example

```
// This example will get a form instance, and log its name
var myForm = Form.GetFormByFORMINSTID(bp, "FORMINSTID");
bp.log0("The form instance name is: " + myForm.Name);
```

GetFormSchema

This API will get an list object containing the form's schema.

Parameters

None.

Returns

List: A list object containing the form's schema.

Example

```
// This example will get a form schema
List mySchema = Form.GetFormSchema();
```

GetInfoMessages

This API will get the list of all current informational messages. The info messages must have been previously added from script.

Parameters

None

Returns

List<FormMessageString>: The list of form error messages.

Example

```csharp
// This example will be called immediately prior to displaying
the form
protected override void BP_Display()
{
    var InfoList = CurrentForm.GetInfoMessages();
    foreach (var info in InfoList)
    {
        bp.log0("INFO MESSAGE: " + info);
    }
}
```

GetJavaScript

This API gets the current blocks of JavaScript that are added to the form.

Parameters

None

Returns

List<string>: The blocks of JavaScript that will be added to the form.

```csharp
// This example will get the Form's Javascript
List myJS = Form.GetJavaScript();
```

Instantiate (Static Method)

This API will instantiate a new form instance. This is an overloaded method with the following possible declarations:

```csharp
public static Form Instantiate(bp BP, string FORMID)
```

```csharp
public   static   Form   Instantiate (bp   BP,   string   FORMID,
                         bool SkipDefaultValues)
```

Parameters

BP: The bp environment

FORMID: The ID of the form definition to instantiate

SkipDefaultValues: Optional Boolean parameter to tell the Form processor to skip setting default form values.

Returns

Form: An instance of the form object, or null if it couldn't be found.

Example

```
// This example will create a new form instance
var NewFormInstance = Form.Instantiate(bp, Con-
tentObject.GetObjectByPathName(bp,
    "Partition1",
    "/My Project/My Form Definition").ID);
bp.log0("The form instance name is: " + NewFormInstance.Name);
```

RecalcFormInstanceName

This API will force the Form engine to recalculate the form instance name. This should be used, for example, if you change a form field value that is used in the instance name programmatically.

Example

```
CurrentForm.RecalcFormInstanceName();
```

RowCount

This API returns the number of rows in an array. The API call will *only* work on an array control.

Parameters

None

Returns

The number of rows in the array.

Examples

```
// Save the number of rows in the array named 'MyArray'
int rows = CurrentForm.FormControl("MyArray").RowCount();
```

SaveAndSubmit

This API will save a form instance (just like SaveForm does), but won't prevent a process from running if the form submission would normally trigger the start of a process.

Parameters

None

Returns

None

Example

```
CurrentForm.SaveAndSubmit();
```

SaveForm

This API will save the current form irrespective of whether it is being displayed. It will increment the form internal version number. If this is a new form instance, calling this API won't automatically start a Process Timeline instance associated with this form definition.

Parameters

None

Returns

None

Example

```
CurrentForm.SaveForm();
```

SynchronizeFields

This API will synchronize all form fields marked as synchronized fields to synchronize with other fields of the same name in the specified process. If no process ID is specified, the process returned by `GetProcess()` is used by default.

Parameters

pPROCINSTID: An optional parameter containing the string value of the Process Instance ID.

Returns

None

Example

```
CurrentForm.SynchronizeFields();
```

UnlockForm

This API will unlock a Form that has been locked for editing.

> ⊗ **Use caution when using this API call! If a form is already locked for editing, and you unlock and edit the form, the locking user may save their version of the form later, and all of your changes will be over-written.**

Parameters

None

Returns

None

Example

```
CurrentForm.UnlockForm();
```

Events

There are six main types of events that are initiated when using Forms, each of which will be discussed separately below. Each of these event types contains a series of individual events that occur in a specific order, called the order of operations. Developers and implementers have access to the order of operations for each event type in two different ways: via scripting, or via the use of Custom Tasks.

When each event fires, scripted event procedures are implemented first, then Custom Tasks associated with the event are implemented. As a result, event scripts can set values or perform other operations that, when the script is complete, are available for the Custom Task to use when it is implemented.

Event Types

New Form is opened

When a new form is opened, the following set of events are fired in the order shown below:

1. BP_FormInitialize Script
2. Form Creation Custom Task
3. BP_ViewStateInit Script
4. View State Init Custom Task
5. BP_Rules Script
6. Before Conditions Custom Task

7. BP_Rules_Post Script
8. After Conditions Custom Task
9. BP_Display Script
10. Form Display Custom Task

Existing Form is opened

When an existing Form is opened, the following set of events are fired in the order shown below:

1. BP_ViewStateInit Script
2. View State Init Custom Task
3. BP_Rules Script
4. Before Conditions Custom Task
5. BP_Rules_Post Script
6. After Conditions Custom Task
7. BP_Display Script
8. Form Display Custom Task

When an event is fired from a form control

When a control event is called, the following set of events are fired in the order shown below:

1. BP_Event Script
2. Event Custom Task
3. BP_Rules Script
4. Before Conditions Custom Task
5. BP_Rules_Post Script
6. After Conditions Custom Task
7. BP_Display Script
8. Form Display Custom Task

When a Form is closed from the OK or task completion button

When a Form is closed by the OK or other task completion button, the following set of events are fired in the order shown below:

1. BP_Event Script
2. Event Custom Task
3. BP_Rules Script
4. Before Conditions Custom Task
5. BP_Rules_Post Script
6. After Conditions Custom Task
7. BP_Validation Script
8. Before Validation Custom Task
9. BP_Validation_Post Script
10. After Validation Custom Task
11. BP_Completed Script
12. Form Completed Custom Task

When a Form is saved without closing

When a Form is saved, but not closed, the following set of events are fired in the order shown below:

1. BP_Event Script
2. Event Custom Task
3. BP_Rules Script
4. Before Conditions Custom Task
5. BP_Rules_Post Script
6. After Conditions Custom Task
7. BP_Display Script
8. Form Display Custom Task

When a Form is saved and closed without completing a task

When a Form is saved and closed without completing a task, the following set of events are fired in the order shown below:

1. BP_Event Script
2. Event Custom Task
3. BP_Rules Script
4. Before Conditions Custom Task

5. BP_Rules_Post Script
6. After Conditions Custom Task

FormControl Class

This object represents a single form field.

> ⓘ **Form array controls are 1-based, so they always start with row number 1 (not 0).**

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Name	String	The name of the form field
FCID	String	The ID of this form field
Text	String	The string/text representation of the form field's value
Value	Data type of Form Field	The raw value of a form field
Number	Decimal	The numerical representation of the form field's value
DateTime	DateTime	The DateTime object that represents the form field's value
DisplayString	String	This property can be used to get (but not set) the form control's display string.
Checked	Boolean	This represents whether the control is checked (true/false, for Check Box controls only)
Expanded	Boolean	This represents whether the control is expanded or collapsed (true/false, for Sections only)

PROPERTY NAME	DATA TYPE	DESCRIPTION
ClientID	String	The client-side ID which can be used by JavaScript to access the control
IsInArray	Boolean	True if the control exists within an array; false otherwise
DDHasItems	Boolean	This represents if the DropDown control has items in it (true/false, for DropDown controls only)
ColumnChildren	List Object	A list of controls in an array column (for Array column controls only)
ArrayFCID	String	If this control is in an array, the FCID of the array control
ArrayNum	Integer	If this control is in an array, the row number
ArrayName	String	If this control is in an array, the name of the array
ArrayControl	Form Control Object	If this control is in an array, the FormControl of the array
ArrayColumns	List Object	If this control is an actual array, the List<FormControl> of children
Display.EventField	Boolean	Sets if the control is an event field (eYNU)
Display.DataType	Code Enum	Sets the Data Type of the control (FormEnums.eDataType enum)
Display.FriendlyName	String	Sets the friendly name of the control

PROPERTY NAME	DATA TYPE	DESCRIPTION
Display.Visible	Boolean	Gets or sets if the control is visible (eYNU)
Display.Enabled	Boolean	Gets or sets if the control is enabled (eYNU)
Display.Required	Boolean	Gets or sets if the control is required (eYNU)
Display.Style	Code Enum	Gets or sets the style of the control
Display.ToolTip	String	Gets or sets the tooltip for this control
Display.Control	Form Control Object	Gets the actual ASP.NET Control of this Form control

Examples

```
// Sets the control "MyTextBox" as an event field
CurrentForm.FormControl("MyTextBox").Display.EventField =
eYNU.Yes;

// Sets the control "MyTextBox" Friendly Name
CurrentForm.FormControl("MyTextBox").Display.FriendlyName =
    "Your current address";
```

Methods

AddRow

This API adds a row to an Array

Parameters

At: (Optional) The position to insert the new row. Without this parameter, the API will add the new row to the end of the Array.

Returns

Integer: The number of rows in the array after the Add

Examples

```
// This will add a row to the beginning of "Array1"
CurrentForm.FormControl("Array1").AddRow(1);

// This will add a row to the end of "Array1"
CurrentForm.FormControl("Array1").AddRow();
```

AddRowToCommentLog

This API adds a row to a comment log

Parameters

CodeComment: (Optional) The CodeComment for the CommentLog. Not all Comment Log controls use a CodeComment. Specify null for these controls.

Date: The DateTime to be associated with the new comment.

UID: The ID of the user to be associated with the new comment.

Comment: The comment to add.

Returns

Integer: The number of rows in the Comment Log after the Add.

Examples

```
// This will add a new comment for the current user
CurrentForm.FormControl("ComLog1").AddRowToCommentLog(null,
    DateTime.Now, CurrentUser.UID, "This is my new comment");
```

AddToDropDown

This API adds an item or list of items to a DropDown control. This is an overloaded method with the following possible declarations:

public void AddToDropDown(IEnumerable<DropDownValue> DDList)

public void AddToDropDown(DropDownValue DDV, params DropDownValue [] DDVs)

public   void   AddToDropDown (IEnumerable<string>   ListText, IEnumerable<string> ListValues)

public void AddToDropDown(IEnumerable<string> Values)

public void AddToDropDown(string pValue, params string[] Values)

Parameters

DDList: DropDownValue objects or strings to add to the dropdown.

Item1, Item2, Item3, etc.: DropDown items to add to the DropDown control.

ListText: List of Text (name) values (use with ValuesList)

ListValues: List of Values to add to the dropdown (use with ListText)

Values: iEnumerable or Params object containing string values

pValue: String value to add

Returns

None

Examples

```
// This example will build a DropDownValue list and a name/-
value pair of lists
var ddList = new List<DropDownValue>();
var textList = new List<string>();
var valList = new List<string>();
for(int i = 0; i < 10; i++)
{
    ddList.Add(new DropDownValue("Entry " + i, i.ToString()));
    textList.Add("Entry " + i);
    valList.Add(i.ToString());
}

// Both of the following AddToDropDown calls will add the same
ten entries
CurrentForm.FormControl("Dropdown1").AddToDropDown(ddList);
CurrentForm.FormControl("Dropdown2").AddToDropDown
(textList,valList);

// The next AddToDropDown call will add only the numerical por-
tion to the
// DropDown control
CurrentForm.FormControl("Dropdown3").AddToDropDown(valList);

// Items: 1, 2, 3, 4, etc.
// Add the entries "Item 1", "Item 2", and "Item 3" to the
// DropDown control
CurrentForm.FormControl("Dropdown4").AddToDropDown("Item 1",
"Item 2",
    "Item 3");
```

ClearDropDown

This API removes all of the items from a DropDown control

Parameters

None

Returns

None

Examples

```
CurrentForm.FormControl("Dropdown1").ClearDropDown();
```

ClearRows

This API removes all of the rows in an Array control

Parameters

None

Returns

None

Examples

```
CurrentForm.FormControl("Array1").ClearRows();
```

ColumnSum

This API adds all of the values in an Array Column control

Parameters

None

Returns

Decimal: The total of each value in the Array Column.

Examples

```
// Sums up each "Price" field in the Array
decimal tot = CurrentForm.FormControl("Price").ColumnSum();
```

FillDropDown

This API adds an item or list of items to a DropDown control, but clears the DropDown first. The API will attempt to re-select the previous value of the

dropdown control, but won't select it if the value no longer exists in the DropDown. This is an overloaded method with the following possible declarations:

`public void FillDropDown(IEnumerable<DropDownValue> DDList)`

`public void FillDropDown(DropDownValue DDV, params DropDownValue [] DDVs)`

`public void FillDropDown (IEnumerable<string> listText, IEnumerable<string> listValues)`

`public void FillDropDown(IEnumerable<string> values)`

`public void FillDropDown(string Value, params string[] Values)`

Parameters

DDList: DropDownValue objects or strings to add to the dropdown.

Item1, Item2, Item3, etc.: DropDown items to add to the DropDown control.

ListText: List of Text (name) values (use with ValuesList)

ListValues: List of Values to add to the dropdown (use with ListText)

Values: iEnumerable or Params object containing string values

Returns

None

Examples

```
// This example will build a DropDownValue list and a name/-
value
// pair of lists
var ddList = new List<DropDownValue>();
var textList = new List<string>();
var valList = new List<string>();
for(int i = 0; i < 10; i++)
{
    ddList.Add(new FormControl.DropDownValue("Entry " + i,
i.ToString()));
    textList.Add("Entry " + i);
    valList.Add(i.ToString());
}

// Both of the following AddToDropDown calls will fill the
DropDown with
// the same ten entries
CurrentForm.FormControl("Dropdown1").FillDropDown(ddList);
CurrentForm.FormControl("Dropdown2").FillDropDown(textList,
```

```
valList);

// The next AddToDropDown call will fill the DropDown with
only the
// numerical portion
// Items: 1, 2, 3, 4, etc.
CurrentForm.FormControl("Dropdown3").FillDropDown(valList);

// Fill the DropDown with the entries "Item 1", "Item 2", and
"Item 3"
CurrentForm.FormControl("Dropdown4").FillDropDown("Item 1",
"Item 2",
    "Item 3");
```

RemoveRow

This API removes a row in an Array

Parameters

At: (Optional) The position of the row to remove. Without this parameter, the API will remove the last row in the Array.

Returns

Integer: The number of rows in the array after the Remove.

Examples

```
// This will remove the first row in "Array1"
CurrentForm.FormControl("Array1").RemoveRow(1);

// This will remove the last row in "Array1"
CurrentForm.FormControl("Array1").RemoveRow();
```

SelectDropDown

This API selects a value in the DropDown control. The API may add the value if it doesn't find it, depending on the parameters. This is an overloaded method with the following possible declarations:

```
public void SelectDropDown(string Value)

public void SelectDropDown(string Value, string Text)

public void SelectDropDown(string Value, bool AddIfNotFound)

public void SelectDropDown(List<string> Values)
```

Parameters

AddIfNotFound: (Optional) Whether or not to add an entry in the DropDown if the API can't find the value (true/false).

Text: (Optional) The text to add if the API can't find the value.

Value: The value to select in the DropDown.

Values: String list of values to select.

Returns

None

Examples

```
// This example will try to select the value "1", and will add
// "[1]"/"1" (name/value) to the DropDown if the API can't find
// "1" already in the DropDown
CurrentForm.FormControl("Dropdown1").SelectDropDown("1");

// This example will try to select the value "1", and will add
// "[Item 1]"/"1" (name/value) to the DropDown if the API can't find
// "1" already in the DropDown
CurrentForm.FormControl("Dropdown2").SelectDropDown("1", "Item 1");

// This example will try to select the value "1", and won't add
// anything to the DropDown if the API can't find "1" already in
// the DropDown
CurrentForm.FormControl("Dropdown3").SelectDropDown("1", false);
```

SetValue

This function sets the value, though not necessarily the display string, of a form control.

Parameters

Val: The value for the form control.

sDisplay: (Optional) The string the form control will display

Example

```
CurrentForm.FormControl("ControlName").SetValue("Value");
```

Sort

This API sorts the rows in an Array. This is an overloaded method with the following possible declarations:

```
public bool Sort(string PrimaryColumn)

public bool Sort(string PrimaryColumn, string SecondaryColumn)

public bool Sort(string PrimaryColumn, string SecondaryColumn,
                 string TertiaryColumn)

public bool Sort(string PrimaryColumn, bool Descending)

public bool Sort(string PrimaryColumn, string SecondaryColumn,
                 bool Descending)

public bool Sort(string PrimaryColumn, string SecondaryColumn,
                 string TertiaryColumn, bool Descending)
```

Parameters

PrimaryColumn: The name of the column which is the first sort key

SecondaryColumn: The optional name of the column which is the second sort key

TertiaryColumn: The optional name of the column which is the third sort key

Descending: Optional Boolean if set true will sort rows descending

Returns

None

Examples

```
// Sort the array
CurrentForm.FormControl("Array1").Sort("ColumnName");
```

SwapRows

This API exchanges the positions of two rows in an Array

Parameters

RowFrom: The first row number to swap

RowTo: The second row number to swap

Returns

None

Examples

```
// Swap the 1st row with the 4th
CurrentForm.FormControl("ArrayName").SwapRows(1, 4);
```

FormMessageString Class

Enables adding a string message to a form.

Properties

PROPERTIES NAME	DATA TYPE	DESCRIPTION
Text	String	Message to use on form.
ControlName	String	Control on form to focus on.
ArrayNum	Integer	If in an array specify the array row number.

Constructor

Parameters

Text: Message to use on form.

ControlName: Control on form to focus on.

ArrayNum: If in an array specify the array row number.

Example

```
var msg = new FormMessageString("Please provide a name.",
"Name");
```

Group Class

This object represents a group of users.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
GID	String	The internal ID of the group
GroupName	String	The unique string to identify the group
Users	List Object	The list of users which belong to the group, returned as a List<User>.

Methods

AddUser

This API adds the specified user to the group. If the user is already in the group, a note of that will be made in the logs, but the function won't add a duplicate user to the group. This is an overloaded method with the following possible declarations:

```
public void AddUser(User User)
```

```
public void AddUser(string UID)
```

Parameters

UID: The ID of the user to add to the group

User: The actual user object to add to the group

Returns

None

Example

```
var oUser = User.GetUserByUserID(bp, "User 1");
var oGroup = Group.GetGroupByName(bp, "Group 1");

// The following two calls do the same thing
// (add "User 1" to "Group 1")
oGroup.AddUser(oUser); // Call with the User object
oGroup.AddUser(oUser.UID); // Call with the User ID
```

CreateGroup

This API creates a user group.

Parameters

BP: The bp environment

Name: The name of the group to create

Returns

None

Example

```
var oGroup = Group.CreateGroup(bp, "Group 1");
```

Delete

This API will remove the Group from the system.

Parameters

None

Returns

None

Example

```
var oGroup = Group.GetGroupByName(bp, "Group 1");
oGroup.Delete();
```

DeleteGroup

This API will remove the Group from the system.

Parameters

BP: The BP Logix environment.

GroupName: The string name of the group to delete.

Returns

Boolean: Returns false of the operation fails.

Example

```
bool deleted = DeleteGroup(bp, "GroupName");
```

GetAllGroups (Static Method)

This static method returns a list of all groups on the system.

Parameters

BP: The bp environment

Returns

List<Group>: A Group object representation of the group

Example

```
var allGroups = Group.GetAllGroups(bp);
```

GetGroupByID (Static Method)

This API will get a group object from the specified ID.

Parameters

BP: The bp environment

ID: The ID of the group to retrieve

Returns

Group: A Group object representation of the group

Example

```
// Normally not used directly
var oGroup = Group.GetGroupByID(bp, "1234");
```

GetGroupByName (Static Method)

This API will get a group object from the specified name.

Parameters

BP: The bp environment

Name: The name of the group to retrieve

Returns

Group: A Group object representation of the group

Example

```
// Get the group named "Group 1"
var oGroup = Group.GetGroupByName(bp, "Group 1");

// Get a list of the users in "Group 1"
var Users = oGroup.Users;
```

HasUser

This API checks if the specified user exists in the group. In some situations, it may be more efficient to view the list of groups to which a specific user belongs, using the User.Groups property. This is an overloaded method with the following possible declarations:

```
public bool HasUser(User User)
```

```
public bool HasUser(string UID)
```

Parameters

UID: The ID of the user to test in the group

User: The actual user object to test in the group

Returns

True/False: Whether or not the specified user exists in the group

Example

```
var oUser = User.GetUserByUserID(bp, "User 1");
var oGroup = Group.GetGroupByName(bp, "Group 1");

// The following two conditional tests test the same thing
// (is "User 1" in "Group 1")
if(oGroup.HasUser(oUser))
{
    // Do action
}
// OR
if(oGroup.HasUser(oUser.UID))
{
    // Do action
}
```

NormalizeGroupList

This API will convert a comma-separated string or DataItem containing a list of Groups into a normalized List object. This is an overloaded method with the following possible declarations:

```
public static List<Group> NormalizeGroupList(bp BP, string pGroupList)
```

```
public static List<Group> NormalizeGroupList(bp BP, string pGroupList,
```

```
                                                    bool
ReturnNullOnInvalid)
public static List<Group> NormalizeGroupList(bp BP, ref DataItem
pGroups)
public static List<Group> NormalizeGroupList(bp BP, ref DataItem
pGroups,
                                                    bool
ReturnNullOnInvalid)
```

Parameters

BP: The BP Logix environment.

pGroupList: A string containing a comma-separated list of groups.

pGroups: A DataItem object containing a list of groups.

ReturnNullOnInvalid: A boolean value specifying whether to return a null List object if the list is invalid.

Returns

List: A List object containing the groups.

Example

```
var groupList = NormalizeGroupList(bp,
"Group1,Group2,Group3");
```

RemoveUser

This API will remove the specified user from the group. This is an overloaded method with the following possible declarations:

```
public void RemoveUser(User User)
public void RemoveUser(string UID)
```

Parameters

UID: The ID of the user to remove from the group

User: The actual user object to remove from the group

Returns

None

Example

```
var oUser = User.GetUserByUserID(bp, "User 1");
var oGroup = Group.GetGroupByName(bp, "Group 1");
oGroup.RemoveUser(oUser);
```

MetaCategory Class

This object represents a MetaData Category.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
PID	String	The Partition ID.
Name	String	TheName of the Meta Data category.
CATID	String	The ID of the Meta Data category.

Methods

GetCategory (Static Method)

This API will return a MetaCategory Object for a specified Meta Data Category.

Parameters

BP: The Process Director BP Object

PID: The Partition ID of the partition containing the Meta Data Category.

CATID: A string containing the Name or CATID of the Meta Data Category.

Returns

MetaCategory: A MetaCategory object.

Example

```
MetaCategory myCat = GetCategory(bp, "PID", "CategoryName");
```

GetCategoryByID (Static Method)

This API will return a MetaCategory Object for a specified Meta Data Category identified by CATID.

Parameters

BP: The Process Director BP Object

PID: The Partition ID of the partition containing the Meta Data Category.

CATID: A string containing the Name or CATID of the Meta Data Category.

Returns

MetaCategory: A MetaCategory object.

Example

```
MetaCategory myCat = GetCategoryByID(bp, "PID", "CATID");
```

GetCategoryByName (Static Method)

This API will return a MetaCategory Object for a specified Meta Data Category identified by Category Name.

Parameters

BP: The Process Director BP Object

PID: The Partition ID of the partition containing the Meta Data Category.

Name: A string containing the Name or Name of the Meta Data Category.

Returns

MetaCategory: A MetaCategory object.

Example

```
MetaCategory myCat = GetCategoryByName(bp, "PID", "Cat-
egoryName");
```

GetCategoryID (Static Method)

This API will accept either a Category name or CATID and return a string containing the CATID for the specified Category, or "" if the Category isn't found.

Parameters

BP: The Process Director BP Object

PID: The Partition ID of the partition containing the Meta Data Category.

CATID: A string containing the Name or CATID of the Meta Data Category.

Returns

String: The CATID for the specified Category, or "" if the Category isn't found.

Example

```
string myCat = GetCategoryID(bp, "PID", "CategoryName");
```

GetRootCategory (Static Method)

This API will return the root-level MetaCategory Object for a specified Partition.

Parameters

BP: The Process Director BP Object

PID: The Partition ID of the partition.

Returns

MetaCategory: A MetaCategory object.

Example

```
MetaCategory myCat = GetRootCategory (bp, "PID");
```

Partition Class

This object represents a Partition.

When developing Form scripts (in the various callback methods such as BP_Event), you are automatically given an instance of the "current" Partition with the CurrentPartition variable.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
PID	String	The internal Partition ID
Name	String	The name of this partition
Description	String	The description of this partition

Methods

CreateFolder

This API will create a folder structure in a Partition

Parameters

PathName: The complete path of the folder structure to create.

Returns

Folder: The folder object created.

Example

```
var NewFolder = CurrentPartition.CreateFolder("/folder-
1/folder2");
```

GetDataSources

This API will return a list of DataSource objects.

Parameters

Returns

List< DataSource>: A list of DataSource objects.

Example

```
var DataSources = CurrentPartition.GetDataSources();
```

GetPartition (Static Method)

This API will get a partition from either a name or ID. This is an overloaded method with the following possible declarations:

```
public static string GetPartitionID(bp BP, string PID)
```

```
public static Partition GetPartition(bp BP, string PartitionID)
```

```
public static Partition GetPartitionByID(bp BP, string Par-
titionID)
```

Parameters

BP: The bp environment.

PartitionID: The Name or ID of the partition to retrieve.

PID: String partition ID of the partition to retrieve.

Returns

Partition: Will return null if partition isn't found.

Example

```
var oPartition = Partition.GetPartition(bp, "PID");
```

GetPartitionByID (Static Method)

This API will get a partition object from the specified ID.

Parameters

BP: The bp environment.

PID: The ID of the partition to retrieve.

Returns

Partition: Will return null if partition isn't found.

Example

```
// Normally not used directly
var oPartition = Partition.GetPartitionByID(bp, "PID");
```

GetPartitionByName (Static Method)

This API will get a partition object from the specified Partition Name.

Parameters

BP: The bp environment.

PartitionName: The ID of the partition to retrieve.

Returns

Partition: Will return null if partition isn't found.

Example

```
var oPartition = Partition.GetPartitionByName(bp, "Par-
titionName");
```

GetPartitionID (Static Method)

This API will get a partition ID from the specified Partition Name.

Parameters

BP: The bp environment.

PID: The Name or ID of the partition to retrieve.

Returns

String: The ID of the Partition or "" if not found.

Example

```
Var oPID = Partition.GetPartitionID(bp, "PartitionName");
```

GetRootFolder

This API will return the ContentObject of the root folder for the partition

Parameters

Returns

ContentObject: The root folder ContentObject or null if the operation fails.

Example

```
var RootFolder = CurrentPartition.GetRootFolder();
```

PDF Class

This object represents a PDF document object.

Methods

CreatePDFForDocument (Static Method)

Creates a reviewable PDF for a specified document. This is an overloaded method with the following possible declarations:

```
public static bool CreatePDFForDocument(bp BP, Document Document)
public static bool CreatePDFForDocument(bp BP, Document Document,
                                        bool OnlyIfNeeded)
```

Parameters

BP: The BP Logix Object.

pDocument: The Document object to convert to PDF.

OnlyIfNeeded: Booloean flag to instruct the system to only create the conversion if needed, i.e., If there is already a web viewable object and you only want to convert documents without an existing PDF object.

Returns

Boolean: Returns false if the operation fails.

Example

```
var oDocument = Document.GetDocumentbyDID(bp, "DID");
CreatePDFForDocument(bp, oDocument);
```

CreatePDFFromDoc (Static Method)

Creates a PDF file from a specified document.

Parameters

BP: The BP Logix Object.

WordPath: String file path where the Word document is stored.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
CreatePDFFromDocument(bp, "C://File/Path/DocumentName.docx",
"C://File/Path/DocumentName.pdf");
```

CreatePDFFromDocument (Static Method)

Creates a PDF file from a specified document.

Parameters

BP: The BP Logix Object.

Document: The Document object to convert to PDF.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
var oDocument = Document.GetDocumentbyDID(bp, "DID");
CreatePDFFromDocument(bp, oDocument, "C://File/Path/Docu-
mentName.pdf");
```

CreatePDFFromForm (Static Method)

Creates a PDF file from a specified Form.

Parameters

BP: The BP Logix Object.

FORMINSTID: The string ID of the Form instance.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
CreatePDFFromForm(bp, "FORMINSTID",
    "C://File/Path/DocumentName.pdf");
```

CreatePDFFromImage (Static Method)

Creates a PDF file from a specified image file. Valid Image formats include GIF, JPG, and PNG.

Parameters

BP: The BP Logix Object.

ImagePath: The string file path to the location of the image to convert.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
CreatePDFFromImage(bp, "C://File/Path/ImageName.png",
    "C://File/Path/DocumentName.pdf");
```

CreatePDFFromRoutingSlip (Static Method)

Creates a PDF file from a Routing Slip for a specified process.

Parameters

BP: The BP Logix Object.

PROCINSTID: The string ID of the Process instance.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
CreatePDFFromRoutingSlip(bp, "FORMINSTID",
    "C://File/Path/DocumentName.pdf");
```

CreatePDFFromString (Static Method)

Creates a PDF file from a given text string.

Parameters

BP: The BP Logix Object.

Text: The string to convert.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
// Create the string to convert
StringBuilder sb = new StringBuilder();
sb.Append("This is the string. ");
sb.Append("This string will be converted to PDF.");

// Convert the string to PDF
CreatePDFFromString(bp, sb.ToString(),
    "C://File/Path/DocumentName.pdf");
```

CreatePDFFromTextFile (Static Method)

Creates a PDF file from a specified text file.

Parameters

BP: The BP Logix Object.

TextPath: The string file path to the location of the text file to convert.

PDFPath: String file path where the new PDF will be stored.

Returns

Boolean: Returns false if the operation fails.

Example

```
CreatePDFFromTextFile(bp, "C://File/Path/textfile.txt",
      "C://File/Path/DocumentName.pdf");
```

CreatePDFFromURL (Static Method)

Creates a PDF file from a web page displayed at a specified URL.

Parameters

BP: The BP Logix Object.

TextPath: The string file path to the location of the text file to convert.

URL: String containing the fully qualified URL of a web page/document.

Returns

Boolean: Returns false if the operation fails.

Example

```
CreatePDFFromURL(bp, "http://www.domain.com/folder/page.htm",
      "C://File/Path/DocumentName.pdf");
```

GetFormFields (Static Method)

Retrieves all of the form fields from a PDF document.

Parameters

BP: The BP Logix Object.

PDFPath: The string file path to the location of the PDF file.

(out) pFormFieldNames: List object that will contain the returned PDF form fields.

Returns

Boolean: Returns false if the operation fails.

List: The List object containing all of the form fields in the PDF document.

Example

```
//The list to store the controls
List<string> myControls = new List<string>();

//Get the form fields to fill the list
GetFormFields(bp, "C://File/Path/DocumentName.pdf",
myControls);
```

MergePDFs (Static Method)

Merges multiple PDF files into a single PDF file. This is an overloaded method with the following possible declarations:

```
public    static    bool    MergePDFs (bp    BP,    IEnumerable<string>
InputPDFPaths,
                          string OutputPDFPath)
public    static    bool    MergePDFs (bp    BP,    IEnumerable<string>
InputPDFPaths,
                          string OutputPDFPath, bool DeleteIn-
puts)
```

Parameters

BP: The BP Logix Object.

InputPDFPaths: The IEnumerable object, such as a List, that contains the file paths of the PDF files to merge.

OutputPDFPath: The file path for the location of the merged PDF.

DeleteInputs: Booloean flag to instruct the system to delete the original PDF files after they have been merged.

Returns

Boolean: Returns false if the operation fails.

Example

```
// The list to store the PDF locations
List<string> MyFiles = new List<string>();
MyFiles.Add("C://File/Path/Document1Name.pdf");
MyFiles.Add("C://File/Path/Document2Name.pdf");

// Merge the PDFs
MergePDFs(bp, MyFiles,
"C://File/Path/MergedDocumentName.pdf");
```

SetFormFields (Static Method)

Sets all of the form fields in a specified PDF document.

Parameters

BP: The BP Logix Object.

PDFPath: The string file path to the location of the PDF file.

FieldNameValues: Dictionary object that will contain field Name/Value pairs to fill out a PDF Form.

TemplatePDFPath: String file path of the Template PDF containing the form fields.

OutputPDFPath: String file path of the PDF output file containing the completed form after the values are entered.

Returns

Boolean: Returns false if the operation fails.

List: The List object containing all of the form fields in the PDF document.

Example

```
// The Dictionary to store the controls
Dictionary<string, string> myFields = new Dictionary<string,
string>();
myFields.Add("Field1", "Value1");
myFields.Add("Field2", "Value2");
myFields.Add("Field3", "Value3");

// Create the filled PDF form from the template
SetFormFields(bp, myFields, "C://File/Path/Template.pdf",
    "C://File/Path/Output.pdf");
```

Process Class

This object represents a Process (definition and/or instance).

This object is derived from the ContentObject class. All properties and methods from the ContentObject are supported for this object, plus the properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Attachments	List Object	Attached files or documents for the process, returned as a List<Document Object>.
DefaultForm	Form Object	Default form definition for the process
DefaultFORMINSTID	String	Default form instance ID for the process

PROPERTY NAME	DATA TYPE	DESCRIPTION
DueTime	DateTime	The date/time the process instance is due
EndTime	DateTime	The date/time the process instance ended
Error	Boolean	Whether the process is in an error state
Initiator	User Object	Process initiator
InstanceName	DefaultForm	Name of the process instance
InstanceObject	Content List Object	The optional content object of this process instance
InstID	String	ID of the process instance
ParentInstanceObject	Content List Object	The optional content object of the parent process instance
Priority	Integer	The priority of the process instance
RunningTasks	List Object	List of running tasks in the process, returned as List<Task>
StartTime	DateTime	The date/time the process instance started
Status	Integer	Status of the process instance, specified in the nStatus field of tblProjectInstance. Please see the table definition of tblProjectInstance for status codes.
TermReason	Integer	If the process instance has

PROPERTY NAME	DATA TYPE	DESCRIPTION
		ended, the reason. Please see the Classes topic for a list of termination reasons and their associated integer code.

Methods

AddToProcess

Adds a ContentObject object to a Process. This is an overloaded method with the following possible declarations:

```
public bool AddToProcess(ContentObject pObj)

public bool AddToProcess(ContentObject pObj, string pGroup)

public bool AddToProcess(string pID)

public bool AddToProcess(string pID, string pGroup)

public bool AddToProcess(string pID, ObjectType pType)

public bool AddToProcess(string pID, ObjectType pType, string pGroup)
```

Parameters

pObj: A Process Director ContentObject object.

pGroup: The string name of a Group to which the object should be added.

pID: The string ID of the Object.

pType: The ObjectType of the Object to be added.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set the Process and content object
var oProcess = Project.GetProcessByID(bp, "PRID");
var oObject = Form.GetFormByFORMID("FORMID");

// Add the object to the process
oProcess.AddToProcess(oObject);
```

Cancel

This API will cancel a running Process Instance.

Parameters

None.

Returns

Boolean: True if operation succeeds

Example

```
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
oProcessInst.Cancel();
```

CheckForAdvance

This API call will force the internal logic for checking for notifications that need to be sent out and steps that need to be transitioned.

Parameters

None

Returns

None

Examples

```
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
bp.log0(oProcessInst.CheckForAdvance());
```

ConvertSysVarsInString (Static Method)

This API converts system variables in a string.

Parameters

String: The string in which to find the system variables.

pDefaultEncode: The BP SysVar Encode method.

Returns

String: The resultant string after converting every system variable.

Examples

```
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
bp.log0(oProcessInst.ConvertSysVarsInString("Created on
{CREATE_DATE}"));
```

GetChildren

This API will get all children of the Process Instance.

Parameters

ObjectType: Optional filter of object types to return.

MapType: Optional filter of map types to return.

GroupName: Optional filter of items in a Group to return.

Returns

List<ContentObject>: List of ContentObjects or null if the operation fails.

Example

```
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
// Gets all items in the group named "Group 1"
var MyChildren = oProcessInst.GetChildren("Group 1");
```

GetProcessByID

This API will return a process object from the specified ID.

Parameters

BP: The bp environment.

pID: The ID of the object to retrieve.

Returns

ProcessObject: Will return null if object isn't found.

Example

```
// Normally not used directly
var oObject = Project.GetProcessByID(bp, "PRID" );
```

GetProcessByInstID

This API will return a process object from the specified ID.

Parameters

BP: The bp environment.

pInstID: The ID of the process instance object to retrieve.

Returns

ProcessInstanceObject: Will return null if object isn't found.

Example

```
// Normally not used directly
var oObject = Process.GetProcessByInstID(bp, "PRINSTID" );
```

GetSubProcesses

This API will return all sub-processes.

Parameters

None

Returns

Process: List of sub processes.

Example

```
var SubProcess = CurrentProject.GetSubProcesses();
foreach (var sub in SubProcess)
{
    bp.log0("Sub process: " + sub.Name);
}
```

GetTaskByName

This API will get the specified ProcessTask.

Parameters

pTaskName: The name of the Process Task to get.

Returns

ProcessTask: The actual ProcessTask or null if the operation fails.

Example

```
var oProcess = Project.GetProcessByID(bp, "PRID");
var Approve = oProcess.GetTaskByName("TaskName");
```

Instantiate

This API will write the instantiate a Process Definition. After instantiating the Process, you can optionally add items to the Process Instance (using Add). Finally, you must call the Start method to actually run the instantiated Process. This is an overloaded method with the following possible declarations:

```
public bool Instantiate()
```

```
public bool Instantiate(Process procParent)
```

```
public bool Instantiate(string InitiatorUID)
```

```
public bool Instantiate(string InitiatorUID, Process procParent)
```

Parameters

procParent: The name of the Process Task to get.

InitatorUID: The User ID of the process initiator.

Returns

Boolean: True if the operation succeeds.

Example

```
var oProcess = Project.GetProcessByID(bp, "PRID");
oProcess.Instantiate();
// Add items to Timeline package here
oProcess.Start();
```

PostEvent

This API will post an event to a process. This is an overloaded method with the following possible declarations:

```
 public  static  bool  PostEvent(bp  BP,  string  PRINSTID,  string
EventName)
```

```
public virtual bool PostEvent(string EventName)
```

Parameters

BP: The BP Logix environment.

PRINSTID: The ProcessInstanceID of the process.

EventName: The string name of the event to post.

Returns

Boolean: True if the operation succeeds.

Example

```
PostEvent(bp, "PRID", "EventName");
```

RecalcInstanceName

This API will generate a new name for a specific Process Instance object.

Parameters

None.

Returns

None.

Example

```
// Re-generates the name of the Process Instance
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
oProcessInst.RecalcInstanceName();
```

ReStart

This API will start a Process instance which has previously completed. You can optionally pass the ProcessTask or TASKID of the step to start. If neither is passed, the Process will restart at the beginning. This is an overloaded method with the following possible declarations:

```
public bool ReStart()
```

```
public bool ReStart(string pTASKID)
```

```
public bool ReStart(ProcessTask pTask)
```

Parameters

pTaskID: The string ID of the task to restart.

pTask: The actual task object to restart.

Returns

Boolean: True if the operation succeeds.

Example

```
// Restarts the Process instance
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
oProcessInst.ReStart();
```

Run

This API will instantiate and start a process definition in one step. You can option-ally add a single item to the Process. This is an overloaded method with the fol-lowing possible declarations:

```
public bool Run()
public bool Run(string pID, ObjectType pType)
public bool Run(string pID, ObjectType pType, string pGroup)
public bool Run(ContentObject pObj)
public bool Run(ContentObject pObj, string pGroup)
```

Parameters

Group: Optional Group of the object to add to the process.

pID: Optional ID of the object to add to the process.

pObj: Optional ContentObject to run.

pType: Optional Type of the object to add to the process.

Returns

Boolean: True if the operation succeeds.

Example

```
var oProcess = Project.GetProcessByID(bp, "PRID");
oProcess.Run();
```

SetCurrentFormInstance

This API will set the current Form Instance (the default Form Instance which will be used to complete tasks) for a Process Instance. The Process must already contain a reference to the Form Instance. This is an overloaded method with the following possible declarations:

```
public bool SetCurrentFormInstance(ContentObject pObj)
public bool SetCurrentFormInstance(string pFORMINSTID)
```

Parameters

FORMINSTID: The form instance ID to make the current.

pObj: The ContentObject to make current.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set the specified form instance as the current form
instance
var oProcessInst = Project.GetProcessByInstID(bp, "PRINSTID");
oProcessInst.SetCurrentFormInstance("FORMINSTID");
```

SetPriority

This API will set the priority of a Process Instance.

Parameters

pPriority: The integer priority to set.

Returns

Boolean: True if the operation succeeds.

Example

```
var oProcessInst = Process.GetProcessByInstID("PRINSTID");
oProcessInst.SetPriority(1);
```

Start

This API will start a Process that has previously been instantiated.

Parameters

None.

Returns

Boolean: True if the operation succeeds.

Example

```
var oProcess = Project.GetProcessByID(bp, "PRID");
oProcess.Instantiate();
// Add items to process package here
oProcess.Start();
```

StartTask

This API will start a Timeline Activity in a running Process.

Parameters

RestartTask: An object variable of the ProcessTask type.

Returns

Boolean: True if the operation succeeds.

Examples

Project Timeline:

```
var oTask = CurrentProject.GetTaskByName("ActivityName");
CurrentProject.StartTask(oTask);
```

ProcessTask Class

This object represents a Process Task.

Properties

The ProcessTask Class is the base class for the WorkflowStep and ProjectActivity classes. It implements the following properties.

PROPERTY NAME	DATA TYPE	DESCRIPTION
PROCID	String	The ID of the process
ID	String	The ID of the current Process Task
TASKID	String	The ID of the Process Task
Name	String	The name of the Process Task
Instructions	String	The instructions for users in this Process Task
Description	String	The description of this Process Task
PROCINSTID	String	The ID of the process instance
TASKINSTID	String	The ID of the Process Task instance
ProcessInstance	Process Instance Object	The Process class for the Process Task
Users	List Object	The list of ProcessTaskUser rep-

PROPERTY NAME	DATA TYPE	DESCRIPTION
		resenting the users in this Process Task, returned as List<Users>
Status	Integer	The status of the step, specified in the nStatus field of tblProjectInstance. Please see the table definition of tblProjectInstance for status codes.
ReAuthenticate	Boolean	Boolean indicating whether reauthorization is required for this process task.
Start	DateTime	The time the Timeline Activity started
End	DateTime	The time the Timeline Activity ended
Due	DateTime	The time the Timeline Activity is due
ShowSignatureComments	Boolean	Boolean indicating whether signature comments are required to complete this process task.
AllowEmailComplete	Boolean	Boolean indicating whether this process task can be completed by email
BaseForm	For Object	The base Form used by this Process Task
InstID	String	The ID of the Process Task instance
EndReason	Integer	The reason the Process task was terminated. Please see the Classes topic for a list of termination reasons and their associated integer code.

PROPERTY NAME	DATA TYPE	DESCRIPTION
Duration	Timespan	The duration of the Process task
Results	List Object	Returns a List object containing the Process task results, returned as a List<String>

Methods

AddUsersToTask

This API will add users to a Process Task. This method can be declared using the following overloads:

```
public bool AddUsersToTask(params User[] pUsers)
```

```
public bool AddUsersToTask(IEnumerable<User> pUsers)
```

```
public bool AddUsersToTask(string pUIDList)
```

Parameters

pUsers: A list of IEnumerable<User> or params objects to add.

pUIDList: A string of UIDs to add (separated by commas).

Returns

Boolean: True if the operation succeeds.

Example

```
// Add the process initiator to the current Process Step
    bool bAddUsers =
    CurrentProcessTask.AddUsersToTask
(CurrentProject.Initiator);
```

CancelTask

This API will set the message property of the Process Task.

Parameters

CancelUID: A string containing the UID of the user for whom to cancel the Process Task.

CancelComments: An string value containing the cancellation comments.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set the message of the Process Task
bool bCancel = CurrentProcessTask.CancelTask("USERUID",
"This is unnecessary");
```

GetProcessTaskByTASKINSTID (Static Method)

This API will get a Process Task instance object from the specified ID.

Parameters

BP: The bp environment.

TASKINSTID: The ID of the Process Task instance to retrieve.

Returns

ProcessTask: Will return null if Process Task isn't found.

Example

```
// Normally not used directly
var oTaskInst = ProcessTask.GetProcessTaskByTASKINSTID( bp,
"TASKINSTID" );
```

GetProcessTaskByTASKID (Static Method)

This API will get a Process Task definition object from the specified ID.

Parameters

BP: The bp environment.

TASKID: The ID of the Process Task to retrieve.

Returns

ProcessTask: Will return null if Process Task isn't found.

Example

```
// Normally not used directly
var oTaskDef = ProcessTask.GetProcessTaskByTASKID( bp,
"TASKID" );
```

GetProcessTasksByPRID (Static Method)

This API will return a list of Process Tasks for a specified process when given a Process ID as a parameter.

Parameters

BP: The BP Logix environment.

pRID: The Process ID of the Process to retrieve.

Returns

List<ProcessTask>: A list object of the Tasks in the specified Process, or null if not found.

Example

```
// Normally not used directly
var oTasks = ProcessTask.GetProcessTasksByPRID( bp, "PRID" );
```

GetTaskByName (Static Method)

This API will get a Process Task object from the specified Name.

Parameters

BP: The BP Logix environment.

Process: The Process object.

TaskName: The name of the Task to get.

Returns

ProcessTask: Will return null if Process Task isn't found.

Example

```
var oObject = Project.GetProcessByID(bp, "PRID" );
var oTask =
    ProcessTask.GetTaskByName( bp, oObject , "Approve Doc" );
```

RemoveUsersFromTask

This API will remove users from a process task. This is an overloaded method with the following possible declarations:

`public bool RemoveUsersFromTask(params User[] pUsers)`

`public bool RemoveUsersFromTask(IEnumerable<User> pUsers)`

`public bool RemoveUsersFromTask(string pUIDList)`

Parameters

pUsers: A list of IEnumerable<User> or params objects to add.

UIDList: A string of UIDs to add (separated by commas).

Returns

Boolean: True if operation succeeds.

Example

```
// Remove the process initiator from the current Activity
CurrentProjectActivity.RemoveUsersFromTask(Cur-
rentProject.Initiator);
```

ResendTaskEmails

This API will resend the task email to all task users.

Parameters

UIDAdmin: Optional string parameter of the user you want to associate with the resend in the audit logs.

Comments: The optional string comments for a completed task.

Returns

ProcessTask: The ProcessTask object or null if the operation fails.

Example

```
ProcessTask.ResendTaskEmails("UIDAdmin", "Comments");
```

Restart

This API will restart a specified task. This is an overloaded method with the following possible declarations:

```
public bool Restart()
```

```
public bool Restart(string UID, string Comments)
```

Parameters

UID: The UID string value of the user for whom the task should be restarted.

Comments: A string containing comments to append to the task that is restarted.

Returns

Returns 'true' if the operation succeeds.

Example

```
// Restart the Process Task
bool bRestart = CurrentProcessTask.Restart();
```

SetDueDate

This API will set the Due Date of a Process Task.

Parameters

DUEDATE: The date to set.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set DueDate of current Activity
CurrentProjectActivity.SetDueDate(DateTime.Now.AddHours(2));
```

SetDuration

This API will set the Duration of a Timeline-based Process Task.

Parameters

SECONDS: The number of seconds to set the duration to. Set to 0 to have the system automatically calculate the duration.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set duration of current Activity
CurrentProjectActivity.SetDuration(10000);
```

SetError

This API will set the message property of the Process Task. This is an overloaded method with the following possible declarations:

```
public bool SetError(string pFormat, params object[] pParams)
```

```
public bool SetError(string strError, bool prepend)
```

Parameters

pFormat: A format string for the error message.

pParams: An params object containing a list of message parameters.

strError: A string value containing the error message to apply to the Process Task.

Prepend: A Boolean value signifying whether the error message should be pre-pended to an existing message, or replace the existing message.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set the error message to display
bool bError = CurrentProcessTask.SetError("Error Message",
true);
```

SetMessage

This API will set the message property of the Process Task. This is an overloaded method with the following possible declarations:

```
public bool SetMessage(string pFormat, params object[] pParams)
public bool SetMessage(string strMessage, bool prepend)
```

Parameters

pFormat: A formatting string.

pParams: A paramsobject containing a list of message parameters.

strMessage: A string value containing the message to apply to the Process Task.

Prepend: A Boolean value determining whether the message should be pre-pended to an existing message, or replace the existing message.

Returns

Boolean: True if the operation succeeds.

Example

```
// Set the message of the Process Task
bool bMessage = CurrentProcessTask.SetMessage("Message text",
true);
```

ProcessTaskUser Class

This object represents a user in a Process Task.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Comment	String	The optional comments if the user completed this Task
End	DateTime	The time that the user completed this Task
ProcessInstance	Process Instance Object	The Process Instance object for the Process containing this Task User
ProcessTaskInstance	Process Task Instance Object	The Process Task Instance object for the Task containing this Task User
PROCID	String	The ID of the Process Definition
Start	DateTime	The time that the user started in this Task
Status	Integer	The status of this user in this Process Task. Please see the table definition of tblProjActivityUserInst for status codes.
SubTaskName	String	The optional name of the sub task assigned to this Task User
TASKID	String	The ID of the Process Task
TASKINSTID	String	The ID of the Task Instance
TASKUID	String	The ID of the Process Task User

PROPERTY NAME	DATA TYPE	DESCRIPTION
TASKUINSTID	String	The ID of the Task User Instance
TermReason	Integer	The termination reason for this user in this Task. Please see the Classes topic for a list of termination reasons and their associated integer code.
TLID	String	The optional task list ID
User	User Object	The user object

Methods

CancelUser

This API will cancel the task for the particular user which this object represents. This is an overloaded method with the following possible declarations:

`public bool CancelUser()`

`public bool CancelUser(string Comments)`

`public bool CancelUser(string Comments, string UIDAdmin)`

Parameters

Comments: (optional) Comments to add to the completion action.

UIDAdmin: (optional) The UID of the (administrative) User who authorized the Task's cancellation.

Returns

Boolean: True if the operation succeeds.

Example

```
// Cancel the User's Task, adding a comment
CurrentProjectActivityUser.CancelUser("Programmatically can-
celing user");
```

CompleteUser

This API will complete the task for the particular user which this object represents. This is an overloaded method with the following possible declarations:

```
public bool CompleteUser()
```

```
public bool CompleteUser(string Selection)
```

```
public bool CompleteUser(string Selection, string Comments)
```

Parameters

Selection: (optional) The branch to take on completion.

Comments: (optional) Comments to add to the completion action.

Returns

Boolean: True if the operation succeeds.

Example

```
// Complete the User's Task, taking the "Approve" branch
CurrentProjectActivityUser.CompleteUser("Approve");
```

ConvertSysVarsInString

This API will convert a SysVar string in the context of the ProcessTaskUser. This is an overloaded method with the following possible declarations:

```
public virtual string ConvertSysVarsInString(string pString)
```

```
public  virtual  string  ConvertSysVarsInString (string  pString,
                                    bp.SysVarEncode pDe-
faultEncode)
```

Parameters

String: The SysVar string to convert.

pDefaultEncode: A BP SysVarEncode object that defines the encoding.

Returns

String: The expansion of the SysVar string

Example

```
// Create the string to append the current object name to the
// string "Name: "
var strNew =
    CurrentProjectActivityUser.ConvertSysVarsInString("Name:
{OBJ_NAME}");
```

GetProcessTaskUserByTASKUID

This API will return a ProcessTaskUser for a Given TaskUserID.

Parameters

BP: The BP Logix environment.

pTASKUID: The string TaskUserID for the user.

Returns

ProcessTaskUser: The ProcessTaskUser object or null if the operation fails.

Example

```
var oUser = ProcessTaskUser.GetProcessTaskUserByTASKUID(bp,
"TASKUID");
```

GetProcessTaskUserByTASKUINSTID

This API will return a ProcessTaskUser for a Given TaskUserInstanceID.

Parameters

BP: The BP Logix environment.

pTASKUINSTID: The string TaskUserInstanceID for the user.

Returns

ProcessTaskUser: The ProcessTaskUser object or null if the operation fails.

Example

```
var oUser =
    ProcessTaskUser.GetProcessTaskUserByTASKUINSTID(bp,
"TASKUINSTID");
```

ResendEmailForUserTask

This API will resend the task email to a specific user.

Parameters

UIDAdmin: Optional string parameter of the user you want to associate with the resend in the audit logs.

Comments: The optional string comments for a completed task.

Returns

ProcessTaskUser: The ProcessTaskUser object or null if the operation fails.

Example

```
var oUser =
    ProcessTaskUser.GetProcessTaskUserByTASKUINSTID(bp,
    "TASKUINSTID");
oUSer.ResendEmailForUserTask("UIDAdmin", "Comments");
```

TaskUsersInTask (Static Method)

This API will return a collection of all Process Task User objects from the specified Task ID.

Parameters

BP: The bp environment.

TASKINSTID: The ID of the Process Task instance with the Process Task Users to retrieve.

Returns

IEnumerable<ProcessTaskUser>: The list of Process Task User objects for the particular Task.

Example

```
// Normally not used directly
var TaskUsers = ProcessTaskUser.TaskUsersInTask(bp,
"TASKINSTID");
```

Project (Process Timeline) Class

This class defines a Process Timeline object (definition or instance).

When developing Form scripts or Timeline scripts, you are automatically given an instance of the current Process Timeline with the CurrentProject variable.

This object extends the Process class, which, in turn, extends the ContentObject class. Thus all properties and methods from the Process and ContentObject classes are available in the Project class, in addition to those below.

Properties

PRID: The ID of the timeline definition.

PRINSTID: The ID of the timeline instance.

Methods

AddToProject

This method attaches an object to the process instance.

Parameters

ContentObject pObj: The content object to be attached to the project.

String pGroup (optional): The group name to use for the object.

String pID: The ID of the object to attach.

ObjectType pType (optional): The ObjectType of the attached object.

String pGroup (optional): The group name to use for the object.

Returns

Boolean: True if the operation succeeds.

Example

```
var oObject = Process.GetProcessByInstID(bp, "INSTID");
var document - Document.GetDocumentbyDID(bp, "DID");
bool success = oObject.AddToProject(document, "Group");
```

This method also has an overloaded method that can be called via the following parameters

GetActivityByName

This method returns a ProjectActivity object (Process Timeline Activity), given the name of the activity.

Parameters

String pActivity: Name of the requested activity.

Returns

ProjectActivity: The requested project activity.

Example

```
var oObject = Process.GetProcessByID(bp, "PID");
var someActivity = oObject.GetActivityByName("ActivityName");
```

GetProjectByPRID

This method returns a project object given the project's ID.

Parameters

bp BP (only necessary if this is a static call): The bp environment.

String PRID: The ID of the timeline definition.

Example

```
var someProject = Project.GetProjectByPRID(bp, "PID");
```

Returns

Project: The project object with the specified ID.

GetProjectByPRINSTID

This method returns a project object given the project instance's ID.

Parameters

String PRINSTID: The ID of the requested project instance.

Returns

Project: The project object whose instance ID matches the one requested.

Example

```
var someProject = Project.GetProjectByPRINSTID(bp,
"PRINSTID");
```

PostEvent (static method)

This method posts an event to the specified project instance.

Parameters

bp BP: The bp environment.

String PRINSTID: ID of the project instance to post an event to.

String EventName: Name of the event to post.

Returns

Boolean: True if the operation succeeds.

Example

```
bool success = Project.PostEvent(bp, "PRINSTID", "EventName");
```

Rollback

This method rolls a process back to a specified activity name.

Parameters

bp BP: The bp environment.

String ActivityName: The activity name to which to roll back the process.

Returns

Boolean: True if the operation succeeds.

Example

```
bool success = CurrentProject.Rollback(bp, "ActivityName");
```

ProjectActivity Class

This object represents a Process Timeline Activity. It is derived from the ProcessTask Class and includes all of its properties, plus the additional properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
ACTID	String	The ID of the Project activity
ACTINSTID	String	The ID of the Activity Instance
ActivityType	Integer	The type of activity, defined by the enum: public enum ActivityType Possible Activity Types are as follows: <table><tr><th>VALUE</th><th>EXPLANATION</th></tr><tr><td>0</td><td>Not set</td></tr><tr><td>1</td><td>User: Users are assigned to activity</td></tr><tr><td>2</td><td>Notify: Notify users</td></tr><tr><td>3</td><td>Process: A Work-</td></tr></table>

PROPERTY NAME	DATA TYPE	DESCRIPTION	
		VALUE	**EXPLANATION**
			flow or Process Timeline is run for this activity
		4	Script: A script is run for this activity
		5	Custom Task: A Custom Task is run for this activity
		6	Meta Data: Used to set the package to this meta data (with some options to copy form instance meta data)
		7	Form: Used to attach a form and/or set the current form viewer for this project
		8	Branch activity
		9	Parent activity
		10	End Timeline
		11	Wait: Wait for time or condition
ActName	String	The name of the Project Activity	

PROPERTY NAME	DATA TYPE	DESCRIPTION
PRID	String	The ID of the Project Definition
PRINSTID	String	The Project Instance ID

Methods

GetProjectActivityByACTID

This API will get a Project/Timeline Activity object from the specified instance ID.

Parameters

BP: The bp environment.

ACTID: The ID of the Project Activity to retrieve.

Returns

ProjectActivity: Will return null if the activity instance isn't found.

Example

```
// Normally not used directly
var oProjectActivity =
    ProjectActivity.GetProjectActivityByACTID (bp, "ACTID");
```

GetProjectActivityByACTINSTID

This API will get a Project/Timeline Activity instance object from the specified instance ID.

Parameters

BP: The bp environment.

ACTINSTID: The ID of the Project Activity instance to retrieve.

Returns

ActivityInstance: Will return null if the activity instance isn't found.

Example

```
// Normally not used directly
var oProjectActivityInst =
    ProjectActivity.GetProjectActivityByACTINSTID (bp,
"ACTINSTID");
```

GetProjectActivityByName

This API will get a Project/Timeline Activity object from the specified Name.

Parameters

BP: The bp environment.

ActivityName: The string name of the Project Activity to retrieve.

Returns

ProjectActivity: Will return null if the activity instance isn't found.

Example

```
// Normally not used directly
var oProjectActivity =
    ProjectActivity.GetProjectActivityByName (bp, "Activ-
ityName");
```

Restart

This API will restart a Timeline Activity.

Parameters

None.

Returns

Returns 'true' if the operation succeeds.

Example

```
// Restart the Process Task
bool bRestart = CurrentActivity.Restart();
```

SetInstanceOwnerDelegate

This method enables the Owner Shared Delegate to be changed on a running activity instance.

Parameters

String pActivityName: The activity name to which to set the Delegation Owner(s).

String pNewOwnerUIDs: A comma-separated list of UIDs to set the new owner(s) of the delegation instance.

Returns

Boolean: True if the operation succeeds.

Example

```
bool success = SetInstanceOwnerDelegate("Activity Name",
"UID1,UID2,UID3");
```

ProjectActivityUser Class

This object represents a user in a Process Timeline Activity (Process Task).

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
ACTID	String	The ID of the Project activity
ACTINSTID	String	The ID of the Activity Instance
ACTUID	String	The ID of the Activity User
ACTUINSTID	String	The ID of the Activity User Instance
IconNumber	String	The icon associated with the chosen result
PRID	String	The ID of the Project Definition
PRINSTID	String	The Project Instance ID
ResultName	String	Result of the activity
RTID	String	ID of the activity result
SubTaskName	String	The optional name of the sub task assigned to this Task User

Methods

CancelUser

This API will cancel the task for the particular user which this object represents

Parameters

Comments: (optional) Comments to add to the completion action.

Admin: (optional) The UID of the (administrative) User who authorized the Task's cancellation.

Returns

Boolean: True if the operation succeeds.

Example

```
// Cancel the User's Task, adding a comment
CurrentProjectActivityUser.CancelUser("Programmatically can-
celing user");
```

CompleteUser

This API will complete the task for the particular user which this object represents

Parameters

Selection: (optional) The branch to take on completion.

Comments: (optional) Comments to add to the completion action.

Returns

Boolean: True if the operation succeeds.

Example

```
// Complete the User's Task, taking the "Approve" branch
CurrentProjectActivityUser.CompleteUser("Approve");
```

ConvertSysVarsInString

This API will convert a SysVar string in the context of the ProcessTaskUser.

Parameters

String: The SysVar string to convert.

Returns

String: The expansion of the SysVar string.

Example

```
// Create the string to append the current object name to the
// string "Name: "
var strNew =
    CurrentProjectActivityUser.ConvertSysVarsInString("Name:
{OBJ_NAME}");
```

Report Class

This object represents a Report object. It is available only to installations that use the Advanced Reporting component

Methods

ExportReport

Exports a report to a file. This is an overloaded method with the following possible declarations:

```
public   static   bool   ExportReport (bp   BP,   string   RID,
                                       string            ExportName,
                                       string         ContentParentID,
                                       string        ContentFolderPath,
                                       string LocalFolderPath)
public   static   bool   ExportReport (bp   BP,   string   RID,
                                       string            ExportName,
                                       string         ContentParentID,
                                       string        ContentFolderPath,
                                       string          LocalFolderPath,
                                       List<NameValue> Variables)
```

Parameters

BP: The BP Logix Object.

RID: The string value of the Report ID.

ExportName: The String name of the exported report.

ContentParentID: The string value of the Content ID of the report's parent object.

ContentFolderPath: The string value of the Report's content path in Process Director.

LocalFolderPath: The string value of the local file path where the exported report will be saved.

Variables: A List object containing the required report variables.

Returns

Boolean: True of the operation succeeds.

Example

```
bool rExport = ExportReport(bp, "ReportID", "ReportName",
    "ContentParentID", "Partition\Reports\Path",
    "C:\\File\Path")
```

Rule Class

This object represents a Business Rule.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Group	String	The string Group name of the Business Rule.

In addition to Properties, Business Rules use the SystemVariableContext object to specify the various context properties for the Business Rule. Please see the [SystemVariableContext topic](#) for more information.

Methods

Evaluate

This API will call the evaluation of the Business Rule, returning the appropriate data for that rule. This is an overloaded method with the following possible declarations:

`public string Evaluate()`

`public string Evaluate(IEnumerable<NameValue> Variables)`

`public string Evaluate(SysVarClass.IContextReadonly Context)`

`public string Evaluate(SystemVariableContextReadonly Context)`

`public string Evaluate (SystemVariableContextReadonly Context, IEnumerable<NameValue> Variables)`

Parameters

Variables: An IEnumerable object containing the variables to pass to the Business Rule.

Context: A SystemVariableContext object containing the context settings for the Business Rule.

Returns

String: The result of the Business Rule's evaluation.

Example

```
var oRule = Rule.GetRuleByID(bp, "RULEID");
string result = oRule.Evaluate();
```

GetRuleByID (Static Method)

This API will get a Business Rule object from the specified ID.

Parameters

BP: The bp environment.

RuleID: The string ID of the partition to retrieve.

Returns

Rule Object: Will return null if the Business Rule isn't found.

Example

```
// Normally not used directly
var oRule = Rule.GetRuleByID(bp, "RULEID");
```

GetRuleByName (Static Method)

This API will return a Business Rule object from the specified Rule Name. This is an overloaded method with the following possible declarations:

```
public static Rule GetRuleByName(bp BP, string PID, string RuleName)
```

```
public static Rule GetRuleByName(bp BP, string PID, string RuleName, string Group)
```

Parameters

BP: The bp environment.

PID: The ID of the partition on which the Business Rule is located.

RuleName: The string name of the Business Rule.

Group: The string Group name of the Business Rule.

Returns

Rule: Will return null if partition isn't found.

Example

```
var oRule = Rule.GetRuleByName(bp, "PartitionID", "RuleName");
```

SetRuleGroup

This API will set a Group name for a Business Rule object.

Parameters

Group: The string Group name to set.

Returns

Rule Object: Will return null if the Business Rule isn't found.

Example

```
// Normally not used directly
var oRule = Rule.GetRuleByID(bp, "RULEID");
oRule.SetRuleGroup("GroupName");
```

SystemVariable Class

This object represents a SystemVariable object.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Type	Code Enum	An enumerated System Variable type.
Data	DataItem Object	A DataItem object containing the System Variable's data.
Parameters	Collections Object	A Collections object containing the Parameters for the System Variable.

In addition to Properties, System Variables use the SystemVariableContext object to specify the various context properties for the System Variable. Please see the SystemVariableContext topic for more information.

Methods

Copy

This API will create a copy of a System Variable.

Parameters

None

Returns

Rule Object: Will return a new copy of the specified System variable.

Example

```
var sv = new SystemVariable(bp.eSysVar.FormField,
    new DataItem(FormEnums.eDataType.String)
    { String = "UserPicker1" });
sv.Parameters["format"] = "UID";
var sEval = sv.Copy();
```

Evaluate

This API will call the evaluation of the System Variable.

Parameters

Context: A SystemVariableContext object containing the context settings for the Condition Set.

Returns

String: The result of the System Variable's evaluation.

Example

```
var sv = new SystemVariable(bp.eSysVar.FormField,
    new DataItem(FormEnums.eDataType.String)
    { String = "UserPicker1" });
sv.Parameters["format"] = "UID";
var sEval = sv.Evaluate();
```

SystemVariableContext Class

This object represents a SystemVariableContext object that contains the context properties for a System Variable or other object.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
Object	ContentList object	A ContentObject object to assign as context
Form	Form Object	A Form object to assign as context
FormControl	Form Control Object	A FormControl object to assign as context
Process	Process Object	A Process object to assign as context
ProcessTask	ProcessTask Object	A ProcessTask object to assign as context
ProcessTaskUser	ProcessTaskUser Object	A ProcessTaskUser object to assign as context
User	User Object	A User object to assign as context

Methods

This class has no methods to call. This class merely stores the context property for a declared SystemVariableContext object. Once the object is declared and properties set, the object can be used to set the context for another Process Director object, such as a Business Rule.

Example

```
// Declare the context object
SystemVariableContext mycontext = new SystemVariableContext();

// Set the context Properties
mycontext.Form = Form.GetFormByFORMID(bp, "FORMID");
mycontext.ProcessTaskUser = Pro-
cessTaskUser.GetProcessTaskUserByTASKUID(bp, "TASKUID");

// Specify a Business Rule to call and apply the context for
evaluation
var oRule = Rule.GetRuleByID(bp, "RULEID");
string result = oRule.Evaluate(mycontext);
```

Task Class

A Task object represents a task assigned to a user. It is distinct from a ProcessTask object, which represents a Process Timeline Activity or Workflow Step. A task object contains such information as the name, description, instruction, and start and due dates of a task.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
UID	String	The UID of the user to whom this task is assigned
FORMINSTID	String	The ID of the form instance this task is related to
FORMID	String	The ID of the form related to this task
TLID	String	This task's ID in the task list
PID	String	ID of the partition this task is on.
PROCINSTID	String	ID of the timeline or Workflow instance to which this task belongs
TASKID	String	ID of the Workflow Step or Timeline Activity to which this task belongs
TASKINSTID	String	ID of the Workflow Step or Timeline Activity instance to which this task belongs
TASKUINSTID	String	ID of the Workflow user step or timeline user activity instance to which this task belongs
tblTaskList	Integer	The database table row related to this task

PROPERTY NAME	DATA TYPE	DESCRIPTION
TaskType	Integer	The type of this task. Please see the table definition of tblTaskList for Task Type codes.
Created	DateTime	The date and time this task was created
Due	DateTime	The date and time this task is due
URL	String	The URL of the Timeline Activity or Workflow task

Methods

CancelTask

This function attempts to cancel the task, and returns a Boolean variable reflecting whether the cancellation succeeded.

Parameters

None

Returns

Boolean: True if the operation succeeds.

Example

```
Task myTask = GetTaskByTLID(bp, "TLID");
myTask.CancelTask();
```

CompleteTask

This function attempts to complete the task given the result, branch taken, or response for the task completion, as well as comments regarding its completion. It returns a Boolean reflecting whether the completion succeeded or not.

Parameters

Selection: The branch taken, response or result for the completion of the task.

Comments: Any comments regarding the completion of the task.

Returns

Boolean: True if the operation succeeds.

Example

```
// Approve the task and complete it
Task myTask = GetTaskByTLID(bp, "TLID");
myTask.CompleteTask("Approved", "Auto-approval via script");
```

GetTaskByTLID (Static Method)

This static function gets a task by the task list ID (TLID).

Parameters

BP: The bp environment.

TLID: The task list ID of the task being requested.

Returns

Task: The task whose TLID matches the one requested.

Example

```
Task myTask = Task.GetTaskByTLID(bp, "TLID");
```

GetTasksForEmail (Static Method)

This static function gets the tasks assigned to a user identified by email address. This is commonly used for unauthenticated users, who are identified solely by email address, and who have no identity in Process Director.

Parameters

BP: The bp environment.

PID or part (optional): The ID or partition object of the partition on which the tasks exist. Is set to null if not specified.

Email: The email address of the user whose tasks are being requested.

Returns

List<Task>: A list of the tasks assigned to this user.

Example

```
List<Task> taskList = new List<Task>();
taskList = GetTasksForEmail(bp, "email@domain.com");
```

GetTasksForUser (Static Method)

This static function gets the tasks assigned to a specified user. This is an overloaded method with the following possible declarations:

```
public static List<Task> GetTasksForUser(bp BP, string PID,
string UID)
```

```
public static List<Task> GetTasksForUser(bp BP, string UID)
```

```
public static List<Task> GetTasksForUser(bp BP, Partition part,
User user)
```

Parameters

BP: The bp environment.

PID or part (optional): The ID or partition object of the partition on which the tasks exist. Is set to null if not specified.

UID or user: The UID or user object of the user whose tasks are being requested.

Returns

List<Task>: A list of the tasks assigned to this user.

Example

```
List<Task> taskList = new List<Task>();
taskList = GetTasksForUser(bp, "UID");
```

User Class

This object represents a user in a Form instance. An instance is a completed Form, or one that is currently being edited.

When developing Form scripts (in the various callback methods such as BP_Event), you are automatically given an instance of the "current" user with the CurrentUser variable.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
AuthType	Code Enum	The type of user. `User.eAuth` is an enum which can return:

PROPERTY NAME	DATA TYPE	DESCRIPTION
		`User.eAuth.Unknown,` `User.eAuth.BuiltIn` `User.eAuth.Windows` `User.eAuth.Windows` `User.eAuth.LDAP` `User.eAuth.External` `User.eAuth.SAML|` `User.eAuth.Header`
AutoDST	Boolean	Is DST enabled for user?
AvgLoginSeconds	Integer	The average number of seconds the user remains logged in.
Company	tblTaskList	The optional company associated with this user
Culture	tblTaskList	A string value containing the User's culture information derived from the .NET CultureInfo Class.
CustomDate	DateTime	Optional custom date/time associated with user
CustomNumber	Decimal	Optional custom number associated with user
CustomString	tblTaskList	Optional custom string associated with user
CustomString2	tblTaskList	Optional second custom string associated with user
Delegate	User Object	A different user to which the user delegates (if null, the user has no delegate)
Dept	tblTaskList	The optional department associated with this user

PROPERTY NAME	DATA TYPE	DESCRIPTION
Description	tblTaskList	The optional description associated with this user
Disabled	Boolean	A Boolean value that returns "true" if the user is disabled.
DisplayString	tblTaskList	Returns either the User-Name if configured, or the UserID
Domain	tblTaskList	Optional domain if user is a Windows user
Email	tblTaskList	Email address of user
ExternalGUID	tblTaskList	The optional external unique ID associated with this user
Groups	List Object	The list of groups to which the user belongs, returned as List<Group>. If you need to check whether a large number of users are in a specific group, it may be more efficient to use the `Group.HasUser ()` function.
ImpersonatedBy	User Object	If the user is being impersonated, this will be a string containing the name of the impersonating user.
LastActivity	DateTime	Timestamp of last activity for user
Locked	Boolean	A Boolean value that returns "true" if the user

PROPERTY NAME	DATA TYPE	DESCRIPTION
		account is locked.
NumLogins	Integer	The number of times the user has logged in.
Office	String	The optional office associated with this user
Phone	String	The optional phone number associated with this user
SessionObjects	Dictionary Object	Returns a dictionary of attributes that can associated with this user while logged in, returned as a `Dictionary<Key,Value>`
Tasks	List Object	Returns the list of Tasks for the user, returned as a `List<task>`
TimeZone	String	Time zone of user
Title	String	The optional title associated with this user
UID	String	The internal ID of the user
UserID	String	The unique string to identify the user
UserName	String	The "friendly" name of the user (for display)

Example

```
// Setting a variable to the current username
var cUser = CurrentUser.UserName;
```

Methods

AddSharedDelegate

This method enables the ability to add or remove a UID from the list of shared delegate users for a user.

Parameters

pGrantAccessToUID: The UID to add to the shared delegation.

Returns

Boolean: True if the operation succeeds.

Example

```
// Add a user to shared delegation
CurrentUser.AddSharedDelegate("UID");
```

AddToGroup

This method adds the user to the specified group. This is an overloaded method with the following possible declarations:

```
public void AddToGroup(Group Group)
```

```
public void AddToGroup(string GID)
```

Parameters

GID: The ID of the group to which to add the user.

Group: The actual group object to which to add the user.

Returns

None

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
var oGroup = Group.GetGroupByName(bp, "GroupName");

// The following two calls do the same thing (add "oUser" to
"oGroup")
oUser.AddToGroup(oGroup); // Call with the Group object
oUser.AddToGroup(oGroup.GID; // Call with the Group ID
```

AddUserToPartition

This method adds the user to the specified partition.

Parameters

Partition: The Partition object to which to add the user.

Returns

Boolean: True if the operation succeeds.

Example

```
// Create a user, and force the user to change the password
after login
var oUser = User.CreateUser(bp, "NewUID", "new@acme.com", "New
User",
    "TEMP_PASSWORD",true);

// Add the new user to a partition
oUser.AddUserToPartition(Partition.GetPartition(bp, "Partition
Name"));
```

AddUserToProfile

This method adds the user to the specified profile.

Parameters

Profile: The name of the profile to which to add the user.

Returns

Boolean: True if the operation succeeds.

Example

```
// Create a user, and force the user to change the password
after login
var oUser = User.CreateUser(bp, "NewUID", "new@acme.com",
    "New User", "TEMP_PASSWORD",true);

// Add the new user to a profile
oUser.AddUserToProfile("ProfileName");
```

CreateUser (Static Method)

This method will create a user object for the specified ID.

Parameters

BP: The bp environment.

UserID: The User ID to create.

Email: The email address of the new user.

UserName: The name of the new user.

Password: The initial password of the new user.

MustChangePassword: If set to true, the new user must change the password after logging in.

AuthType: Added in v5.44.500, this parameter takes one of the User.eAuth enum types to specify the authorization type for the user.

Returns

User: A user object representation of the new user.

Example

```
// Create a user, and force the user to change the password
after login
var oUser = User.CreateUser(bp, "NewUID", "new@acme.com",
    "New User", "Temp_Password", true);
```

CreateExternalUser (Static Method)

This method will create a user object for the specified ID. The user object is an external user which can only be signed in via an external authentication system. This is an overloaded method with the following possible declarations:

```
public  static  User  CreateExternalUser (bp  BP,  string  UserID,
                                string Email, string User-
Name)
public  static  User  CreateExternalUser (bp  BP,  string  UserID,
                                string Email, string User-
Name,
                            Defines.eAuth UserType)
 public  static  User  CreateExternalUser (bp  BP,  string  UserID,
                                string Email, string User-
Name,
                            string                   GUID,
                            Defines.eAuth UserType)
 public  static  User  CreateExternalUser (bp  BP,  string  UserID,
string                                         Email,
                            string   UserName,   string
GUID,
```

```
                           string                Domain,
                   Defines.eAuth UserType)
```

Parameters

BP: The bp environment.

UserID: The User ID to create.

Email: The email address of the new user.

UserName: The name of the new user.

UserType: Optional type of user.

GUID: Optional Unique identifier for the user.

Returns

User: A user object representation of the new user.

Example

```
// Create a SAML user
var oUser = User.CreateExternalUser (bp, "NewUID", "new@acme.-
com",
"New User", User.eAuth.SAML);
```

DecodeLDAPName

This method will decode an LDAP name string to only return the "name" part. This is typically used for groups.

Parameters

Name: The LDAP Name string.

Returns

String: The "name" portion of the LDAP string.

Example

```
string name = DecodeLDAPName("cn=ABC");
```

DelegateUser

This method enables a user to delegate all tasks.

Parameters

UID: The UID to delegate to.

Returns

Boolean: True if the operation succeeds.

Example

```
// Delegate current users tasks to another user
CurrentUser.DelegateUser("UID");
```

Delete

This method will remove the User from the system.

Parameters

None

Returns

None

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
oUser.Delete()
```

DeleteUser

This method will remove the User from the system.

> ⛔ **BP Logix strongly recommends that users be disabled, not deleted.**
> **Deleting a user will delete all of their system history as well, including all**
> **process partitipation.**

Parameters

BP: The BP Logix environment.

UserUID: The User UID of the User to delete.

Returns

Boolean: True if the operation succeeds.

Example

```
bool deleted = DeleteUser(bp, "USERUID");
```

DisableUser

This method will disable the specified user's account and cancel the user in any active process tasks.

Parameters

None

Returns

Boolean: True of the operation succeeds.

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
oUser.DisableUser();
```

DisableUserEmail

This method will disable the ability to send email to a specified user.

Parameters

None

Returns

Boolean: True of the operation succeeds.

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
oUser.DisableUserEmail();
```

EnableUser

This method will enable a Process Director User. This is an overloaded method with the following possible declarations:

public bool EnableUser()

public bool EnableUser(bool pEnable) **[Deprecated]**

Parameters

pEnable [Optional]: Setting this parameter to "True" will enable the user, and setting it to "False" will disable the user. The method that uses this parameter has been deprecated, but remains in the product for backward compatibility.

Returns

Boolean: True of the operation succeeds.

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
oUser.EnableUser();
```

EnableUserEmail

This method will enable the ability to send email to a specified user.

Parameters

None

Returns

Boolean: True of the operation succeeds.

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
oUser.EnableUserEmail();
```

GetAllUsers (Static Method)

This static method will get a list of all users on the system.

Parameters

BP: The bp environment.

Returns

List<User>: A list of all Users on the system.

Example

```
var allUsers = User.GetAllUsers(bp);
```

GetUserByEmail (Static Method)

This method will get a user object from the specified email address.

Parameters

BP: The bp environment.

UserEmail: The email address of the user to retrieve.

Returns

User: A user object representation of the user.

Example

```
var oUser = User.GetUserByEmail(bp, "jane@acme.com");
```

GetUserByExtID (Static Method)

This method will get a user object from the specified external ID. For instance, Active Director users user the SID as the unique external identifier.

Parameters

BP: The bp environment.

ExtID: The unique external ID of the user to retrieve.

Returns

User: A user object representation of the user.

Example

```
var oUser = User.GetUserByExtID (bp, user-
principal.Sid.ToString());
```

GetUserByID (Static Method)

This method will get a user object from the specified UserID.

Parameters

BP: The bp environment.

pUID: The UID of the user to retrieve. This is the internal ID used by Process Director.

Returns

User: A user object representation of the user.

Example

```
// Normally not used directly
var oUser = User.GetUserByID(bp, "UID");
```

GetUserByUserID (Static Method)

This method will get a user object from the specified ID. This is an overloaded method with the following possible declarations:

```
public static User GetUserByUserID(bp BP, string pUserID)

public   static   User   GetUserByUserID (bp   BP,   string   pUserID,
                                    Defines.eAuth AuthType)
```

Parameters

BP: The bp environment.

pUserID: The ID of the user to retrieve.

AuthType: An optional `Defines.eAuth` object that defines the user's authorizations, i.e. User.eAuth.Windows, User.eAuth.BuiltIn, etc.

Returns

User: A user object representation of the user.

Example

```
// Normally not used directly
var oUser = User.GetUserByUserID(bp, "UID",
User.eAuth.BuiltIn);
```

GetUserByIDOrUserID (Static Method)

This method will get a user object from the specified UID or UserID. This is an overloaded method with the following possible declarations:

```
public static User GetUserByUserID(bp BP, string pUser)

public   static   User   GetUserByUserID (bp   BP,   string   pUser,
                                    Defines.eAuth AuthType)
```

Parameters

BP: The bp environment.

pUser: The UID or UserID of the user to retrieve.

AuthType: An optional Defines.eAuth object that defines the user's authorizations, i.e. User.eAuth.Windows, User.eAuth.BuiltIn, etc.

Returns

User: A user object representation of the user.

Example

```
// Normally not used directly
var oUser = User.GetUserByUserID(bp, "UID");
```

ImportUsersFromExcel

This method enables you to import a batch of users from an Excel spreadsheet.

Parameters

BP: The BP class.

DID: The Document ID of the Excel file in the Content List.

SheetName: The sheet name in the Excel file that contains the list of users.

Returns

String: The result of the import operation.

Example

```
// Import the users in the "Users.XLSX" file in the sheet
named "Sheet1"
var DID = ContentObject.GetObjectByPathName(bp,
"PartitionName",
    "/Users.XLSX");
var RES = BPLo-
gix.WorkflowDirector.SDK.User.ImportUsersFromExcel(bp,
    DID.ID, "Sheet1");
```

InGroup

This method checks if the user exists in the specified group.

Parameters

GID: The ID of the group in which to test the user.

Group: The actual group object in which to test the user.

Returns

Boolean: Whether or not the user exists in the specified group.

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
var oGroup = Group.GetGroupByName(bp, "GroupName");
// The following two conditionals test the same thing (is
```

```
"oUser"
// in "oGroup")
if(oUser.InGroup(oGroup))
{
    // Do action
}
if(oUser.InGroup(oGroup.GID))
{
    // Do action
}
```

LockUserAccount

This method locks the user account for a specified user.

Parameters

UID: The UID of the user account to lock.

Returns

Boolean: True if the operation succeeds.

Example

```
bool locked = LockUserAccount("UID");
```

NormalizeUserList (Static Method)

This method will take a UID or UserID list in string format or in a DataItem object and convert it to a normalized List object. This is an overloaded method with the following possible declarations:

```
public static List<User> NormalizeUserList(bp BP, string pUser-
List)
public static List<User> NormalizeUserList(bp BP, string pUser-
List,
                                            bool   ReturnNullOnIn-
valid)
public static List<User> NormalizeUserList(bp BP, ref DataItem
pUsers)
public static List<User> NormalizeUserList(bp BP, ref DataItem
pUsers,
```

```
                                              bool    ReturnNullOnIn-
valid)
```

Parameters

BP: The bp environment.

pUserList: A string containing a comma-separated UID or UserID list of users.

pUsers: A DataItem object containing a list of UIDs or UserIDs.

ReturnNullOnInvalid: A boolean value the determine whether to return a null object if the list is invalid. True will return the null on onvalid.

Returns

List: A List object containing the UIDs or UserIDs.

Example

```
// Normally not used directly
List oUsers = NormalizeUserList(bp, "UID1,UID2,UID3");
```

RemoveFromGroup

This method will remove the User the specified group.

Parameters

GID: The ID of the group from which to remove the user.

Group: The actual group object from which to remove the user.

Returns

None

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
var oGroup = Group.GetGroupByName(bp, "GroupName");
oUser.RemoveFromGroup(oGroup);
```

RemoveSharedDelegate

This method will remove a User from a shared delegation.

Parameters

pRemoveUID: The UID of the user to remove from shared delegation.

Group: The actual group object from which to remove the user.

Returns

Boolean: True if the operation succeeds.

Example

```
// Remove a user from shared delegation
CurrentUser.RemoveSharedDelegate("UID");
```

ReplaceWithUser

This method will replace a Process Director user with a specified replacement user.

Parameters

ReplacementUID: The UID of the replacement user.

Returns

Boolean: True of the operation succeeds.

Example

```
var oUser = User.GetUserByUserID(bp, "UID");
oUser.ReplaceWithUser("ReplacementUID");
```

SetCurrentUserContext

This method changes or sets the current user context.

Parameters

BP: The BP Logix environment.

UID: The string UID or UserID of the user.

Returns

Boolean: True if the operation succeeds.

Example

```
bool curr = SetCurrentUserContext("UID");
```

UnDelegateUser

This method enables a user to un-delegate all tasks.

Parameters

None.

Returns

Boolean: True if the operation succeeds.

Example

```
// Un-delegate current users tasks
CurrentUser.UnDelegateUser();
```

UnlockUserAccount

This method unlocks the user account for a specified user.

Parameters

UID: The UID of the user account to unlock.

Returns

Boolean: True if the operation succeeds.

Example

```
bool unlocked = UnlockUserAccount("UID");
```

UpdateLastActivityTime

This method will update the user object so set the current time as the last activity time for that user.

Parameters

None.

Returns

Boolean: True if the operation succeeds.

Example

```
// Update the last activity time for the user
CurrentUser.UpdateLastActivityTime();
```

UpdateUser

This method will update the user object in the database with any changes made to the object's properties (e.g. "UserID", "UserName", etc.)

Parameters

None.

Returns

Boolean: True if the operation succeeds.

Example

```
// Select a user
var oUser = User.GetUserByUserID(bp, "UID");

// Change the user's Company name
oUser.Company= "MyCompany";

// Update the user
oUser.UpdateUser();
```

UserExists

This method enables you to check to see if a user exists.

Parameters

BP: The BP Logix environment.

pUserID: A string UserID.

Optional Parameters

pAuthType: A User.AuthType object specifying the authorization type (Unknown, BuiltIn, Windows, etc.). If this parameter is omitted, then Process Director will default this parameter to User.eAuth.BuiltIn.

Returns

Boolean: True if the user exists.

Example

```
bool exists = UserExists(bp, "UID", User.eAuth.BuiltIn);
```

Workflow Class

> ⊗ **The Workflow object is the legacy process model used in early versions of Process Director. BP Logix recommends the use of the Process Timeline object, and not the Workflow object. The Workflow object remains in the product for backwards compatibility, but doesn't receive any new functionality updates, other than required bug fixes. No new fea-**

tures have been added to this object since Process Director v4.5. All new process-based functionality is solely added to the Process Timeline.

This object represents a Workflow (definition and/or instance).

When developing Form scripts (in the various callback methods such as BP_Event) or Workflow scripts, you are automatically given an instance of the "current" Workflow with the CurrentWorkflow variable.

This object is derived from the Process class, which is, in turn, derived from the ContentObject class. All properties and methods from the ContentObject and Process classes are supported for this object, plus the properties below.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
WFID	String	The ID of the Workflow definition
WFINSTID	String	The ID of the optional Workflow instance

Methods

AddToWorkflow

This API will add an item to a Workflow instance. This is an overloaded method with the following possible declarations:

```
public bool AddToWorkflow(string ID, ObjectType Type)
```

```
public bool AddToWorkflow(string ID, ObjectType Type, string Group)
```

Parameters

ID: ID of the object to add to the Workflow.

Type: Type of the object to add to the Workflow.

Group: Optional Group of the object to add to the Workflow.

Returns

Boolean: True if the operation succeeds.

Example

```
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
oWorkflowInstance.AddToWorkflow("OBJID", ObjectType.Document);
```

Cancel

This API will cancel a running Workflow instance.

Parameters

Returns

Boolean: True if the operation succeeds.

Example

```
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
oWorkflowInstance.Cancel();
```

GetChildren

This API will get all children of the Workflow instance.

Parameters

ObjectType: Optional filter of object types to return.

MapType: Optional filter of map types to return.

GroupName: Optional filter of items in a Group to return.

Returns

List<ContentObject>: List of ContentObjects or null if the operation fails.

Example

```
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
// Gets all Workflow items in a named group
var MyChildren = oWorkflowInstance.GetChildren("GroupName");
```

GetWorkflowStepByName

This API will get the specified WorkflowStep. This is an overloaded method with the following possible declarations:

```
public WorkflowStep GetWorkflowStepByName(string pStepName)
public static WorkflowStep GetWorkflowStepByName(bp BP, Workflow pWorkflow,
                                                 string pStepName)
```

Parameters

BP: The bp environment.

pStepName: The name of the Workflow Step to get.

pWorkFlow: The Workflow object containing the step.

Returns

WorkflowStep: The actual WorkflowStep or null if the operation fails.

Example

```
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
var ApproveStep = oWorkflowInstance.GetWorkflowStepByName
("StepName");
```

GetWorkflowByWFID (Static Method)

This API will get a Workflow definition object from the specified ID.

Parameters

BP: The bp environment.

WFID: The ID of the Workflow definition to retrieve.

Returns

Workflow: Will return null if Workflow isn't found.

Example

```
// Normally not used directly
var oWorkflowDef = Workflow.GetWorkflowByWFID(bp, "WFID");
```

GetWorkflowByWFINSTID (Static Method)

This API will get a Workflow instance object from the specified ID.

Parameters

BP: The bp environment.

WFINSTID: The ID of the Workflow instance to retrieve.

Returns

Workflow: Will return null if Workflow isn't found.

Example

```
// Normally not used directly
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
```

JumpToStep

This API will jump to a specific Workflow Step.

Parameters

JumpFrom: Workflow Step name to jump from.

JumpTo: Workflow Step name to jump to.

Returns

Boolean: True if the operation succeeds.

Example

```
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
// Restarts the Workflow instance at a specific step
oWorkflowInst.JumpToStep("FromStepName","ToStepName");
```

JumpToStepID

This API will jump to a specific Workflow Step.

Parameters

FromSTID: Workflow Step ID to jump from.

ToSTID: Workflow Step ID to jump to.

Returns

Boolean: True if the operation succeeds.

Example

```
var oWorkflowInstance = Workflow.GetWorkflowByWFINSTID(bp,
"WFINSTID");
// Restarts the Workflow instance at a specific step
oWorkflowInst.JumpToStepID("FromSTEPID", "ToSTEPID");
```

PostEvent

This API will Post an event to a Workflow that is waiting on a wait step. This is an overloaded method with the following possible declarations:

`public bool PostEvent(string EventName)`

`public static bool PostEvent(bp BP, string WFINSTID, string EventName)`

Parameters

BP: The bp environment.

WFINSTID: The ID of the Workflow instance to retrieve.

EventName: The name of the event to Post.

Returns

Boolean: True if the operation succeeds.

Example

```
// Wake up a Workflow
Workflow.PostEvent(bp, "WFID", "EventName");
```

ReStart

This API will start a Workflow Step that has previously completed.

Parameters

RestartStep: The WorkFlowStep object of the step to restart.

Returns

Boolean: True if the operation succeeds.

Example

```
// Restarts the Workflow Step
WorkflowStep myStep = GetWorkflowStepByName("StepName");
ReStart(myStep);
```

WorkflowStep Class

> ⊗ **The Workflow object is the legacy process model used in early versions of Process Director. BP Logix recommends the use of the Process Timeline object, and not the Workflow object. The Workflow object remains in the product for backwards compatibility, but doesn't receive any new functionality updates, other than required bug fixes. No new features have been added to this object since Process Director v4.5. All new process-based functionality is solely added to the Process Timeline.**

This object represents a Workflow Step.

This object is derived from the ProcessTask class. All properties and methods from the ProcessTask are supported for this object, plus the properties below.

When developing Form scripts (in the various callback methods such as BP_Event) or Workflow scripts, you are automatically given an instance of the "current" Workflow with the CurrentWorkflowStep variable.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
WFID	String	The ID of the Workflow definition
STID	String	The ID of the Workflow Step
StepName	String	The name of this step
WFINSTID	String	The ID of the Workflow instance
STINSTID	String	The ID of the Workflow Step instance
WfInst	WorkflowInstance Object	The Workflow instance object that contains this step
Message	String	The optional message if

PROPERTY NAME	DATA TYPE	DESCRIPTION
		this step is complete
Error	Boolean	Is the Workflow Step in an error state?

Methods

GetWorkflowStepByName (Static Method)

This API will get a Workflow Step object from the specified Name.

Parameters

BP: The bp environment.

Workflow: The Workflow object.

StepName: The name of the step to get.

Returns

WorkflowStep: Will return null if Workflow Step isn't found.

Example

```
var var oWorkflow = Workflow.GetWorkflowByWFID(bp, "WFID");
var oWorkflowStep = WorkflowStep.GetWorkflowStepByName(bp,
    oMyWorkflow, "StepName");
```

GetWorkflowStepBySTID (Static Method)

This API will get a Workflow Step definition object from the specified ID.

Parameters

BP: The bp environment.

STID: The ID of the Workflow Step to retrieve.

Returns

WorkflowStep: Will return null if Workflow Step isn't found.

Example

```
// Normally not used directly
var oWorkflowStepDef = WorkflowStep.GetWorkflowStepBySTID(bp,
"STID");
```

GetWorkflowStepBySTINSTID (Static Method)

This API will get a Workflow Step instance object from the specified ID.

Parameters

BP: The bp environment.

STINSTID: The ID of the Workflow Step instance to retrieve.

Returns

WorkflowStep: Will return null if Workflow Step isn't found.

Example

```
// Normally not used directly
var oWorkflowStepInst = WorkflowStep.GetWorkflowStepBySTINSTID
(bp, "STINSTID");
```

JumpToStep

This API will set the Workflow's running step from the current (this) step to a different step. This is an overloaded method with the following possible declarations:

```
public bool JumpToStep(string pStepName)
```

```
public bool JumpToStep(string pStepName, string pAdminUID)
```

```
public bool JumpToStep(string pStepName, string pAdminUID, string pAdminComment)
```

Parameters

StepName: The name of the step to which to jump (the destination step).

AdminUID: Optional UID of the user which caused the step jump.

AdminComment: Optional remark on the reason for the jump.

Returns

Boolean: True if the operation succeeds.

Example

```
// Jump from the current step to the "Approve Request" step
CurrentWorkflowStep.JumpToStep("StepName");
```

JumpToStepID

This API will set the Workflow's running step from the current (this) step to a different step. This is an overloaded method with the following possible declarations:

```
public bool JumpToStepID(string pSTID)

public bool JumpToStepID(string pSTID, string pAdminUID)

public bool JumpToStepID(string pSTID, string pAdminUID, string
pAdminComment)
```

Parameters

pSTID: The ID of the step to which to jump (the destination step)

pAdminUID: Optional UID of the user which caused the step jump

pAdminComment: Optional remark on the reason for the jump

Returns

Boolean: True if the operation succeeds.

Example

```
// Jump from the current step a named step
var wfStep = WorkflowStep.GetWorkflowStepByName(bp,
    CurrentWorkflow, "StepName");
CurrentWorkflowStep.JumpToStep(wfStep.STID);
```

WorkflowStepUser Class

> ⓧ **The Workflow object is the legacy process model used in early versions of Process Director. BP Logix recommends the use of the Process Timeline object, and not the Workflow object. The Workflow object remains in the product for backwards compatibility, but doesn't receive any new functionality updates, other than required bug fixes. No new features have been added to this object since Process Director v4.5. All new process-based functionality is solely added to the Process Timeline.**

This object represents a user in a Workflow Step.

When developing Form scripts (in the various callback methods such as BP_Event) or Workflow scripts, you are automatically given an instance of the "current" Workflow User with the CurrentWorkflowStepUser variable.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
WFID	String	The ID of the Workflow definition
STID	String	The ID of the Workflow Step
SUID	String	The ID of the Workflow Step user
TLID	String	The optional task list ID
WFINSTID	String	The ID of the Workflow instance
STINSTID	String	The ID of the step instance
SUINSTID	String	The ID of the step user instance
User	User Object	The user object
Status	Integer	The status of this user in this Workflow Step. Please see the table definition of tblProjActivityUserInst for status codes.
Comment	String	If the user completed this task, the optional comments
Start	DateTime	The time that the user started in this step
End	DateTime	The time that the user completed this step
TermReason	Integer	The termination reason for this user in this step. Please see the Classes topic for a list of termination reasons and their associated integer code.

PROPERTY NAME	DATA TYPE	DESCRIPTION
BranchName	String	The branch the user selected to complete this step
SubTaskName	String	The optional name of the sub task assigned to this Task User

Methods

GetWorkflowStepUserBySUINSTID (Static Method)

This API will get a Workflow Step definition object from the specified ID.

Parameters

BP: The bp environment.

SUINSTID: The ID of the Workflow Step user to retrieve.

Returns

WorkflowStepUser: Will return null if Workflow Step isn't found.

Example

```
// Normally not used directly
var oWorkflowStepUser = Work-
flowStepUser.GetWorkflowStepUserBySUINSTID(bp, "SUINSTID");
```

Restart

This API will restart a Workflow Step for a specified user. There are two overloads for calling this function:

```
WorkflowStep.Restart(string UID)
```

```
WorkflowStep.Restart(string UID, string Comments)
```

Parameters

UID: The UID of the user to restart in the Workflow Step.

Comments: Optional parameter containing the Routing Slip comments to add to the Workflow Step.

Returns

None

Example

```
var oWorkflowStepInstance = Work‐
flowStep.GetWorkflowStepBySTINSTID(bp, "STINSTID");
oWorkflowStepInstance.Restart();
```

Workspace Class

This object represents a Workspace object.

Properties

PROPERTY NAME	DATA TYPE	DESCRIPTION
oPROFILEID	String	The ID of the Workspace
Name	String	The name of the Workspace
Description	String	The description of the Workspace.

Methods

GetWorkspaceByName

This API will return the Workspace object specified by the Name.

Parameters

bp: The Process Director environment.

pName: The string Name of the Workspace.

Returns

A Workspace object.

Example

```
var myWS = Workspace.GetWorkspaceByName(bp, "WorkspaceName");
```

GetWorkspaceByPROFILEID

This API will return the Workspace object specified by the Profile ID.

Parameters

bp: The Process Director environment.

pProfileID: The string Profile ID of the Workspace.

Returns

A Workspace object.

Example

```
var myWS = Workspace.GetWorkspaceByPROFILEID(bp, "PROFILEID");
```

JavaScript APIs

Process Director uses some JavaScript APIs to perform operations to manipulate Form data, or to Open Forms in special circumstances as described below.

Form Data

Process Director enables you to get or set Form Field values through JavaScript. Using the JavaScript API enables you to use the appropriate JavaScript Input field attributes for the field type you wish to set. For instance, the value of a text box can be set using the ".value" attribute, while the value of a checkbox can be set using the "checked" attribute.

When using JavaScript, you can get or set the value of a field using the following syntax:

CurrentForm.FormControls["FieldName"].valueattribute

Examples

```
// Set the Value of a Text Box
CurrentForm.FormControls["Text1"].value = "MyValue";

//Get the value of a text box
var myValue = CurrentForm.FormControls["Text1"].value;

// Set the value of a checkbox
CurrentForm.FormControls["Checkbox1"].checked = true;

// Get the Value of a checkbox
var myValue = CurrentForm.FormControls["Checkbox1"].checked;
```

JavaScript can be used directly on the Form by inserting an HTML control on the Form and erasing the Name property of the HTML control.

iPopupSimple Command

When creating a Workspace for a new application, it's possible to display a Process Director form as the primary portlet for the Workspace. You might wish to do this when you want the workspace to have a highly customized display and layout, rather than displaying the standard Knowledge Views/Task List in the standard workspace portlets. Using a Form in this manner enables you to design a work-space with custom colors, object locations, branding, etc.

There is one drawback to this method of UI design, in that Forms do not call other forms directly in Process Director. In most cases, the purpose of a Form is to collect data about a specific process instance, which can be submitted and used to view/update the data used in the process instance. This Form usage assumes that the primary form will not call additional Forms and, if additional forms are needed, they can be added to the Process Timeline via the **Form Actions** Timeline Activity. Indeed one can switch back and forth between Forms using this Timeline Activity type.

Using a Form as the primary dashboard UI for a Workspace, therefore, represents a unique challenge, in that you will probably want users to have the ability to open Forms, Knowledge Views, or other objects, which is a capability that forms don't have. To enable this feature, a JavaScript command, **iPopupSimple**, enables you to configure a Form button to open a new object instance in a popup window directly from the Form. This JavaScript command takes a single parameter, which is the URL of the Process Director object you wish to open, e.g.:

```
iPopupSimple('URLToOpen');
```

This command is conceptually very simple, but using it in an integration, especially when you'd like to use different buttons to open different Process Director objects, adds a fair amount of complexity. Since that's so, let's take a look at how to implement `iPopupSimple` using an implementation example.

First, the Workspace for our sample application is set to use a single portlet for the workspace that displays the Form we wish to use as the Workspace Dashboard.

In this example, the Submitter Dashboard is a Process Director Form that will display the application UI. When displayed to the end user, it will look like this:
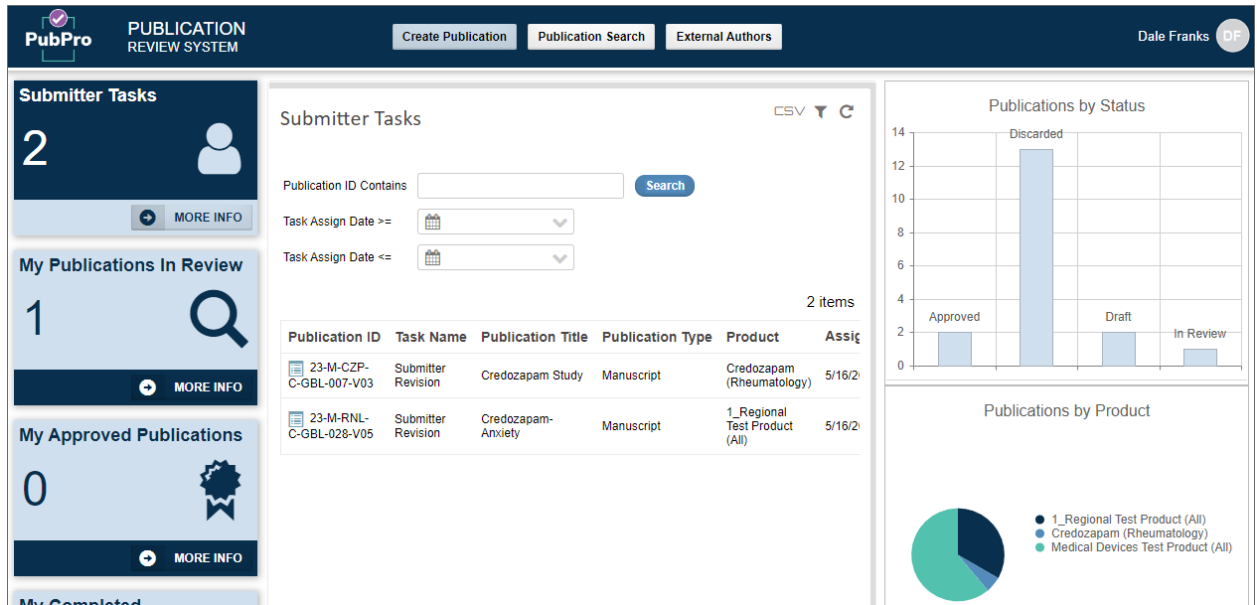
All of the objects the user sees are designed into the Form. There are several buttons on this form, so for the purposes of this example, let's concentrate solely on the CreatePub button at the top of the Form, which is labeled Create Publication. This button must, when clicked, open a new Form that the user can fill out and submit. Let's take a close look at top portion of the Submitter Dashboard Form in the Online Form Designer.
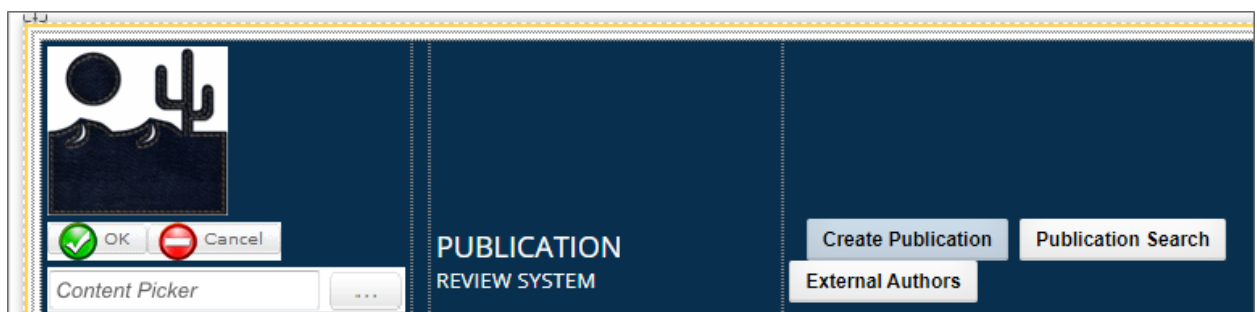
In addition to the CreatePub button, there's also a Content Picker control on the left side of the page, named CreateForm. The CreateForm control enables us to specify the Form we want to open when the CreatePub button is clicked. We do this by setting the Default Value property of the CreateForm control to a Content Item, and choosing the Publication Project Form as the Content Item.

We'll use the value of the CreateForm control to help construct the URL we need to call with the `iPopupSimple` command.

The CreatePub button is a simple Button control. In this case, since we'll be using it to call the `iPopupSimple` JavaScript command, we'll use its OnClientClick property to call `iPopupSimple` by placing the command in that property box.

Normally, clicking a Button control prompts a reload of the Form to perform some sort of server-side action. We don't want to do that in this case, so we're using the OnClientClick property to capture the button click event on the client side, which is to say, in the browser, rather than prompting a server-side event.

The full syntax of the `iPopupSimple` command we're using is:

`iPopupSimple (' {Interface_ URL}form.aspx?formid= {:CreateForm,-format=ID}');`

To shorten the URL we need to provide, the JavaScript URL parameter we're passing uses two system variables. `INTERFACE_URL` provides the base URL of our Process Director Installation, ending in a forward slash, e.g., `HTTPS://pub-lications.bplogix.com/`. The Form System Variable for the CreateForm Content Picker is formatted to return the FormID of the Form we want to open as a new instance. At run-time, this configuration ensures that, when the CreatePub button is clicked, the Publication Project Form is opened in a new popup window, just as Forms normally open in Process Director.

Technically, we don't need to use either of these two system variables to construct the URL parameter that we're passing to `iPopupSimple`. We could simply hard-code the Form URL into the command. But, that would require typing a long, complex URL into the OnClientClick property box. Moreover, should we need to change the Form we want to open with this button, such as after an update to the application, we'd need to come back and re-type a different (long and complex) URL again. With this configuration, if we need to change the Form we want to open, we just have to change the Default Value of the CreateForm Content Picker to a different Form to update the URL parameter automatically.

In this example, we've only discussed opening a Form instance, but using the same process, we can also open Knowledge Views, Charts, or other Content List objects as well. All we need to do is provide the appropriate Content Picker and Button controls, and configure the onClientClick property for each of the buttons we need.

Language/Culture Localization

Customizing the Process Director UI

The default language of Process Director is US English. All built-in system strings are stored in the \Program Files\BP Logix\Process Director\website\App_GlobalResources\Resource.resx file (a standard .NET resource file containing strings). To localize these to another language, create a separate file called Resource.[Culture].resx in the same folder (where [Culture] is the language/culture type, as specified in MSDN.). As an example, a file for German localization would be named "Resource.de.resx". This file will contain the translated strings. You only need to include the strings that have been translated into this file.

At runtime, the system will first try to find a string in the Resource.[Culture].resx file, and then it will try the default Resource.resx file. Since the Resource.resx file is the final fallback file in this hierarchy, this means that you can also create a customized US English resources file to change specific items of the Process Director user interface text by creating a Resource.en-US.resx file. Process Director will check that file first, and use any custom text from that file, before falling back to the main Resource.resx file for the standard UI text items.

It is also possible to completely localize the entire Process Director UI by making a copy of the Resource.resx your master translation file. This is, of course a very, very

large file, and you'd need to also add the custom strings you want to translate to each copy of this file.

> ⛔ **Do not modify the Resource.resx file; it will be overwritten on any upgrade/patch.**

Each user is able to set their own culture/language in their profile, and the system will use the appropriate resx file to display the customized interface for their selected culture.

Once you've edited the RESX files, you'll need to use the [Locales custom variable](#) to add the Cultures to Process Director.

- Resource.en-US.resx for US English
- Resource.it.resx for Italian
- Resource.ja-JP.resx for Japanese
- etc.

See [MSDN](#) for the various "culture strings" (i.e. the portion of the file name between "Resource." And ".resx").

Edit the strings in each new Resource.[culture_string].resx file for the specific language.

Copy all the translated RESX files into the \Program Files\BP Logix\Process Director\website\App_GlobalResources\ folder.

Ensure that the [Culture] portion of the file name for the RESX file matches the *pValue* parameter used in the Locales custom variable.

Form Customization/Localization

In addition to the ability to localize Process Director's user interface, Forms can also be localized so that the same Form can be used for users from different cultures. Localized Forms will automatically display all of the Form field labels in the language appropriate to the user's culture.

Localizing Forms requires two steps:

1. Placing the translations for the Form field labels in the appropriate RESX file for each culture.

2. Using variables for Form labels instead of using fixed text (e.g., First Name, Last Name, etc.).

Let's look at each step separately, along with some best practices to implement when performing each step.

Placing strings in the RESX file

Each time you add a field to a Form, you must create a new culture string for that field's label in the RESX file for each culture. When you do so, it would be wise to create a naming convention for Form Field strings. For example, for a common field, such a "First Name", you might create a resource string name like "FirstName".

In the /website/App_GlobalResources folder of your Process Director installation, create a master string file (a standard .NET RESX file) named "strings.resx" that contains all custom strings that you want to use. This file uses the default language for Process Director, which is US English, so there's no need to specifically create a US English-version RESX file, though you'll need to create a language specific strings file for other languages, e.g., strings.es.resx for Spanish.

Every RESX file must have a resource string entry for the field's label, with a different value in each file. So, if you have a RESX file for US English, Spanish, and German the following entries would appear in each RESX file for the FirstName string:

- **strings.resx (US English):** frmCommonFirstName "First Name"
- **strings.es.resx (Spanish):** frmCommonFirstName "Nombre de Pila"
- **strings.de.resx (German):** frmCommonFirstName "Vorname"

Using variables for Form field labels

You can access your own custom strings using a system variable such as {string:My_String} which will be replaced at run time with the localized string. In the Form template, for each field label you'd like to customize, use a System Variable or Label control, or just type the system variable in plain text.

```
{string:UserSectionHead}

{STRING:FirstName}:
[                              ]

{STRING:LastName}:
[                              ]

{STRING:Department}:
[                              ]
```

Additionally, you can use the comments portion of the RESX file to specifically identify which forms use each string.

Name	Value	Comment
Approve	Genehmigen	Common Result
Department	Abteilung	Form: Common Field
FirstName	Vorname	Form: Common Field
LastName	Familienname	Form: Common Field
Reject	Abweisen	Common result
UserSectionHead	Nutzerinformation	Form: Common Field

In the example above, the implementers can, in addition to the form field labels, also add common activity results, such as "Approve" by using System Variables such as {string:Approve} that will be replaced with the localized version of the string "Approve" for the current user's culture. As you can see in this example, Form fields have a comment that marks them as common Form fields, while "Approve" and "Reject" are listed as common results.

Now, we can see how the Form will display for each culture:

English

User Information

First Name:

Last Name:

Department:

Spanish

Informacion del Usuario

Nombre de Pila:

Apellido:

Departamento:

German

Nutzerinformation

Vorname:

Familienname:

Abteilung:

Customization File

You can customize your Process Director installation by editing the customization file, **vars.cs.ascx,** that's located in the `\Program Files\BP Logix\Process Director\website\custom\` directory of your Process Director installation. This directory is created during the initial installation of the product. The files in this directory are *not* overwritten during subsequent installations, such as when you perform an upgrade to a new version of Process Director.

The customization file is separated into several methods, each of which enables you to customize some aspect of Process Director. Please expand the code example below to see an example of the customization file and its organization.

Code Example

The following are called only for the vars files:

```
// Called BEFORE database initialized
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
}

// Called AFTER database initialized
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Before making SDK calls that access the database,
ensure DB has
    // been opened by checking bp.DBOpenComplete
}

// Called on pages that are "in" a partition
public override void SetPartitionVars(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                BPLogix.WorkflowDirector-
.SDK.Partition Partition)
{
}

/*
// This override allows you to return different values based
on a token
// The default operator simply stores the values in a Dic-
tionary
// Example: bp.Vars["GlobalVar1"] = "My Value";
```

```csharp
public override string this[string pToken]
{
    get
    {
        return "Token value: " + pToken;
    }
    set
    {
    }
}
*/

// Called on every login for customizing the error checking of
a user
public override bool CanLogin(BPLogix.WorkflowDirector.SDK.bp
bp,
                                Login_Struct pLoginStruct,
                                out string pErrorString)
{
    pErrorString = null;

    /*
    if (pLoginStruct.UserID.ToUpper() == "BAD_USER")
    {
        pErrorString = "That is a bad user";
        return false; // Prevent login
    }
    */
    return true; // Allow login
}

// Called to test if user can call specific web service
public override bool CanCallService(BPLo-
gix.WorkflowDirector.SDK.bp bp,
User CurrentUser, string API)
{
    return true;
}

// Called to test if we can disable this user
public override bool CanDisableUser(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                string UID,
                                out string Reason)
{
    Reason = null;
    return true;
}
```

```csharp
// Called after every successful login
public override void LoginComplete(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                Login_Struct LoginStruct,
                                User CurrentUser)
{
}

// Called after every logoff
public override void LogoffComplete(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                        User LogoffUser)
{
}

// Called to return the optional page to navigate to after the
login occurs
public override string GetFirstPage(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                        string pFirstPage)
{
    return base.GetFirstPage(bp, pFirstPage);
}

// Called when a user clicks on "forgot password". In this
callback, you can set a
// new password for the user,
// and you can force the user to change their password after
their first login.
// Return true to allow the password retrieval, or false to
prevent the password retrieval.
// Note that fAllowRetrievePassword must be set to true to
allow any user to
// retrieve their password. Only built-in (vs Active Dir-
ectory) users
// can retrieve their password.
public override bool ForgotPassword(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                        string UserID,
                                        string OldPassword,
                                        out string NewPassword,
                                        out bool MustChangePass-
word)
{
    NewPassword = null; // Do not change old password
    MustChangePassword = false; // Do not force user to change
password after login
    return true;
}
```

```
// Called to validate a password change for a user
// Return true to allow the password change, or false to pre-
vent
// the change. You can set the Reason parameter
// to a string representing the reason for the invalid pass-
word
// (e.g.,, password too short)
// NOTE: This is only enabled with the "Compliance Option"
public override bool ValidatePassword(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                      string UserID,
                                      string OldPassword,
                                      string NewPassword,
                                      out string Reason)
{
    Reason = null;
    /*
    // Example to ensure all new passwords are at least 5 char-
acters long
    if (NewPassword.Length < 5)
    {
        Reason = "password too small";
        return false;
    }
    */
    return true;
}

// Called to customize the string that is displayed for each
user.
// This will set the display string for the entire product.
// NOTE: You must set fAllowCustomUserString to true in order
for this exit to be called
public override void UserDisplayString(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                      ref string UserString,
                                      User UserClass)
{
    /*
    // This will add the custom string to the end of the user
display string
    if (!UserClass.CustomString.bpIsNullOrEmpty())
    {
        UserString = UserString + " / " + User-
Class.CustomString;
    }
    */
}
```

```csharp
// Called to customize the string that is displayed for each
user.
//This will set the display string only in the object that
calls this function.
// NOTE: You must set fAllowCustomUserString to true in order
for this exit to be called
public override void UserDisplayString2(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                        ref string UserString,
                                        User UserClass)

{
}


// This event is called prior to synchronizing a user from Act-
ive Directory
// using the AD Sync Profiles.
// Returning true allows the user to be sync'ed. Returning
false will
// prevent the user from being sync'ed.
public override bool AD_Sync_User(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                          string ProfileName,
                          System.DirectoryServices.AccountManage-
ment.UserPrincipal User)
{
    /*
    // Do not sync the user if the email address is null
    if (string.IsNullOrEmpty(User.EmailAddress))
        return false;
    */
    return true;
}

// This event is called after synchronizing a user from Active
Directory using the AD Sync Profiles.
// This can be used, for example, to modify the user record
after the sync.
public override void AD_Sync_User_Complete(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                          string ProfileName)
                          System.DirectoryServices.AccountMan-
agement.UserPrincipal User,
                          string UID)
{
    /*
    var TempUser =
```

```
BPLogix.WorkflowDirector.SDK.User.GetUserByID(bp, UID);
    if (TempUser != null)
    {
        TempUser.CustomString = "Some custom value";
        TempUser.UpdateUser();
    }
    */
}

// This event is called prior to synchronizing a group from
Active Directory
// using the AD Sync Profiles.
// Returning true allows the group to be sync'ed.
// Returning false will prevent the group from being sync'ed.
public override bool AD_Sync_Group(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                          string ProfileName,
                          System.DirectoryServices.AccountManag-
ement.GroupPrincipal Group)
{
    return true;
}

// This event is called after synchronizing a group from Act-
ive Directory using the AD Sync Profiles.
public override void AD_Sync_Group_Complete(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                     string ProfileName,
                     System.DirectoryServices.AccountManagement.-
GroupPrincipal Group,
                     string GID)
{
    /*
    var TempGroup = BPLo-
gix.WorkflowDirector.SDK.Group.GetGroupByID(bp, GID);
    if (TempGroup != null)
    {
    }
    */
}

// Called when a sync profile starts / end
public override void AD_Sync_Start(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                                   string ProfileName,
                                   string ProfileID,
                                   bool TestMode)

{
}
```

```
public override void AD_Sync_End(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                              string ProfileName,
                              string ProfileID,
                              bool TestMode)

{
}
```

While many different aspects of your installation's operation can be altered via writing your own C# code in the customization file, the primary customization method is to use one or more of the many Custom Variables that are already built into the product, and which provide an easy means of changing the default operation of the system. The built-in Custom Variables are organized into different categories in this documentation, as outlined in the Custom Variables topic.

ⓘ **Access to the customization file is generally limited to On-Premise customers. Customers with Cloud installations can only access their customization file directly via an additional license option that enables them to access the /custom folder via an SFTP connection. Otherwise, Cloud customers can submit a technical support ticket that describes the desired change, and BP Logix will make it for them.**

Form Control Styles

Forms have the ability to set the style for Enabled/Disabled form controls, Required form fields, and form fields in an error state using the "Default Styles" section on the Form properties page. To modify the list of default styles displayed in the dropdown, edit the vars.cs.aspx file.

Code Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Form style choices
    // If you comment out the first line, it will append new
values to the
    // default list
    bp.Vars.StyleOptions.Clear();
```

```
     bp.Vars.StyleOptions.Add("border:3px solid #AD1A10;");
     bp.Vars.StyleOptions.Add("border:3px solid red;");
     bp.Vars.StyleOptions.Add("border:3px solid blue;");
     bp.Vars.StyleOptions.Add("border:1px solid blue;");
     bp.Vars.StyleOptions.Add("border:1px dashed blue;");
     bp.Vars.StyleOptions.Add("border:1px dotted red;");
}
```

Creating Your Own Custom Variables

Process Director enables you to set your own system variables in the vars.cs.ascx file. All custom vars that you create are formatted as strings, so you should use string syntax when setting the variable's value, i.e., use double quotes (`""`) around the value. Process Director will convert custom vars to numbers on the fly if numeric comparisons or calculations are required. Custom vars that you create are stored in a custom dictionary as key/value pairs.

There are two methods available in the custom vars file for creating custom vars: `SetSystemVars` and `PreSetSystemVars`. The `PreSetSystemVars` method is called prior to initializing the Process Director database, while the `SetSystemVars` method is called after database initialization.

In general, this means that the default method to use when creating custom vars is the `PreSetSystemVars` method; however, if you need to set the value of the var based on information stored in the Process Director database, such as the identity of the user or the workspace in which the user is working, you must use the `SetSystemVars` method.

Using the syntax `{CustomVar:MY_VAR}` elsewhere in Process Director will return the custom variable (in this case, `MY_VAR`) value that was defined in the customization file.

Code Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This creates a new custom var that can be accessed from
anywhere
    // within Process Director by using the syntax {Cus-
tomVar:MY_VAR}
    bp.Vars["MY_VAR"] = "My string";
```

```
}
```

Session Variables

Process Director provides the ability to create session variables. These variables exist only for the length of a single user session, and are discarded when the user's session ends, such as when the user logs out of Process Director, or after the specified period of inactivity that is set in IIS. Session variables can be created via custom scripting in a Form or process, or inside one of the sections of the customization file, depending on when it's necessary to create the variable. For instance, if you want to set the session variable any time a page loads in a partition, you could set the session variable in the `SetPartitionVars` method of the customization file.

Code Example

```csharp
public override void SetPartitionVars(BPLo-
gix.WorkflowDirector.SDK.bp bp,
                        BPLogix.WorkflowDirector.SDK.Partitio-
n Partition)
{
    // Default session var
    CurrentUser.SessionObjects["GROUP_ID"] = "";
    var mygroup = BPLo-
gix.WorkflowDirector.SDK.Group.GetGroupByName(bp, "MyGroup");

    // Only want to set var for mygroup & valid user
    if (CurrentUser.InGroup(mygroup))
    {
        // Set the System Variable
        CurrentUser.SessionObjects["GROUP_ID"] =
         bp.ConvertSysVarsInString("{BV:MyGroupData.GROUPID,
         $UserId="+CurrentUser.UID.ToString()+"}");
    }
}
```

Once the session variable has been set, the variable is addressable through code, or in any Process Director Object using the Session System Variable .

Shared Delegation

Shared Delegation can be universally disabled in the Custom Vars file via the `SharedDelegationAllowed` method.

Code Example

```csharp
// We want to disable the shared delegation for the system
public override bool SharedDelegationAllowed
(BPLogix.WorkflowDirector.SDK.bp bp,
                                       Task TaskList,
                                       User CurrentUser,
                                       User
TaskListUser,

                                       out string
ReasonString)
{
    ReasonString = "Reason for disabling";
    // Disable Shared Delegation
    return false;
}
```

Custom Variables

Process Director provides many built-in Custom Variables that control or customize various functions on the system. Most Custom Variables should be set in the `PreSetSystemVars` method of the customization file, but this isn't universally true. The `PreSetSystemVars` method is called **prior** to the system accessing the Process Director database, while `SetSystemVars` is called **after** the database has been initialized. Any custom setting that relies on information contained in the Process Director database, such as the ID of a user or Process Director object, will only work properly when called from the `SetSystemVars` method. Please refer to the documentation's code samples for the system variables you're setting, to ensure you place each Custom Variable in the appropriate method in the customization file.

The built-in Custom Variables are arranged into general subject sections to help make them easier to find in the documentation. To see the Custom Variables for each subject, you can use the Table of Contents displayed on the upper right corner of the page, or click on one of the links below.

Active Directory: Active Directory synchronization variables.

Administration: Sets administrative options for the system.

Auditing: Controls the operation of the audit logs.

Collaborative Features: Controls the operation of the CDM/CDA features (for appropriately licensed installations).

Default Settings: Enable changing the default options for your Process Director Installation.

LDAP: Controls the operation of LDAP synchronizations.

List Maximums: Controls the maximum number of items returned by various system list objects, such as Knowledge Views.

Logs: Controls the operation of the system logging functions.

Miscellaneous: General variables that control various Process Director options/operations.

ML/AI: Controls the operation of Machine Learning/Artifical Intelligence features.

Mobile Application: Controls the operation of the BP Logix Mobile App (for appropriately licensed installations).

Password Enforcement: Controls the complexity and security of user passwords.

Process Administration: Enables altering the default settings for who can access process administration features.

Reporting Tool: Enables altering the default settings of the Advanced Reporting Component.

REST: Customizes the settings associated with REST data usage.

SAML: Configures the connection to a SAML IdP for external authentication (Federated Identity).

Social Media: Sets options for accessing data from Social Media applications.

System: Enables altering the general system properties.

Tasks: Enables customizing the default settings related to user tasks.

User Interface: Controls the appearance of the Process Director User Interface.

User Info SlideOut: Enables altering the operation of the User Info SlideOut that displays their profile to end users.

Users: Controls the operations of various system features relating to users and user accounts.

Active Directory Custom Variables

You can customize some of the ways in which Process Director interacts with Active Directory by editing the custom variables in this section of the documentation.

ADAuthNoDomain

Certain Active Directory installations can't accept a domain as part of the credential validation. This flag can be set to ensure the domain isn't passed to the Active Directory validation.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Do not use the domain on the credential validation
    bp.Vars.ADAuthNoDomain = true;
}
```

ADAuthSettings

This variable enables you to configure specific authentication settings for each domain for validating users at login. The account used for each domain must have permission to open the Active Directory and validate credentials.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Configure specific administrator accounts to use to val-
idate
    //logins for 2 domains.
    bp.Vars.ADAuthSettings.Add(new ADAuthSetting("MY_DOMAIN_
1", "AD_Admin", "pwd1"));
    bp.Vars.ADAuthSettings.Add(new ADAuthSetting("MY_DOMAIN_
2", "AD_Admin", "pwd2"));
}
```

ADGrouphierarchy

This variable enables you to configure the type(s) of groups that the user is a member of when using the Active Directory Synch.

Values

VALUE NAME	DESCRIPTION	DEFAULT
ADGroupHierarchyOptions.AllAuthGroups	This will enable adding the user to every security group	

VALUE NAME	DESCRIPTION	DEFAULT
	(even hierarchical) they are a member of.	
ADGroupHierarchyOptions.AllGroups	This will enable adding the user to every group (security AND distribution) (even hierarchical) they are a member of.	
ADGroupHierarchyOptions.None	setting will only add the users to the groups they are directly a member of.	Default

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // The default
    // This will enable adding the user to every security
group
    // (even hierarchical) of which they are a member.
    bp.Vars.ADGrouphierarchy = ADGroupHi-
erarchyOptions.AllAuthGroups;

    // This will enable adding the user to every group (secur-
ity AND
    // distribution, even hierarchical) of which they are a
member.
    bp.Vars.ADGrouphierarchy = ADGroupHi-
erarchyOptions.AllGroups;

    // This setting will only add the users to the groups of
which they are
    // directly a member
    bp.Vars.ADGrouphierarchy = ADGroupHierarchyOptions.None;
}
```

AD_NormalOptions

Can be used to configure options used in the PrincipalContext to connect to the Active Directory server. See the [Microsoft documentation for ContextOption Enumeration](#) for a description of the options.

The available options are:

- System.DirectoryServices.AccountManagement.ContextOptions.Negotiate
- System.DirectoryServices.AccountManagement.ContextOptions.Signing
- System.DirectoryServices.AccountManagement.ContextOptions.Sealing

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set the flags for options for Active Directory
    bp.Vars.AD_NormalOptions =
     Sys-
tem.Dir-
ectoryServices.AccountManagement.ContextOptions.Negotiate;
}
```

ADSSLOptions

Can be used to configure options used in the PrincipalContext to connect to the Active Directory server for SSL encrypted sessions. See the [Microsoft documentation for ContextOption Enumeration](#) for a description of the options.

The available options are:

- System.DirectoryServices.AccountManagement.ContextOptions.Negotiate
- System.DirectoryServices.AccountManagement.ContextOptions.SecureSocketLayer

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set the flags for SSL options for Active Directory
    bp.Vars.AD_SSLOptions =
     Sys-
tem.Dir-
ect-
oryServices.AccountManagement.ContextOptions.SecureSocketLayer
;
}
```

AD_SyncUsersByGroupRecurse

Can be used to disable group recursion for Active Directory synchronization, if needed. The default value for this variable is "true", and should usually remain so. An issue with Microsoft Windows 2016, however, may cause synchronization to fail when limiting synchronization to a specific user or group. Setting this value to "false" can serve as a workaround for this issue.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable group recursion for Active Directory syn-
chronization
    bp.Vars.AD_SyncUsersByGroupRecurse = false;
}
```

fADSyncAllowManagerOtherOU

This boolean variable, when set to "true" enables you to sync a manager with a user if the manager is synced from a different OU (AD Root Path) than the user. The default value of this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Sync managers from a different OU than the users they
manage
    bp.Vars.fADSyncAllowManagerOtherOU = true;
}
```

fReenableUsersOnSync

This integer variable determines what happens to pre-existing disabled users on an AD Sync. If set, the sync will re-enable these users, but won't if the flag isn't set. The default value of this flag is 'true'.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fReenableUsersOnSync = false;
}
```

fSyncExtraLog

This Boolean variable sets whether to turn on extra, Level 0 logging when a sync is performed. The default value of this variable is False.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fSyncExtraLog = false;
}
```

nMaxUsersToDisableOnSync

This integer variable sets the maximum number of users to disable on an Active Directory Synchronization. The default value of this setting is 30.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.nMaxUsersToDisableOnSync = 30;
}
```

nMaxPercentUsersToDisableOnSync

This double variable sets the maximum percentage of users to disable on an Active Directory Synchronization. The default value of this setting is .10, which is 10%.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.nMaxPercentUsersToDisableOnSync = .10;
}
```

nMinLDAPUsersWithGroupsBeforeDisable

This integer variable sets the minimum number of users that need to have groups before Process Director removes any group memberships during an Active Directory synchronization. The default value of this variable is 10.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.nMinLDAPUsersWithGroupsBeforeDisable = 10;
}
```

Administration Custom Variables

Process Director 's Administrative settings can be customized by editing the custom variables in this section.

Activity Checking Custom Variables

Process Director advances processes and send reminders based on recurring activity checks that occur on the system. These activity checks occur when certain process events are initiated, when the Activity Check page is run, or at minimum intervals specified by default Custom Variable settings.

When the activity check page runs, Process Director will determine which timer processing functions need to occur. (There are a few things like GOALS that will be evaluated EVERY time the activity check runs because they are low-impact.) You can control the timer functions by setting custom variables in the vars.cs customization file. These timer functions are NOT controlled by the frequency that the activity_check.aspx page is scheduled. The following configuration variables control the amount of time to wait between different types of timer processing. The following Custom Variables control the timing of advancing process steps and reminders that are time based:

- **TimerSecondsCheckWfAdvance:** Checks Workflows to advance time-based steps. The default timing is 6 hours.
- **TimerSecondsCheckWfReminders:** Checks Workflows to send time-based reminders. The default timing is 1 hour.
- **TimerSecondsCheckProjAdvance:** Checks Process Timelines to advance time-based Activities. The default timing is 6 hours.
- **TimerSecondsCheckProjReminders:** Checks Process Timelines to send time-based reminders. The default timing is 1 hour.

For Process Director v5.39 and below, the default timings for TimerSecondsCheckWfAdvance and TimerSecondsCheckProjAdvance are set to 2 hours. This was changed in newer versions as the shorter time limits could use excessive system resources on very active systems.

Additionally, the **fDisableUserPrediction** Custom Variable will, when set to "false", enable more accurate prediction of activity times based on each specific user that has been assigned to an activity. *This setting will consume more system resources when being used*, so the default value for this variable is "true". When set to "true", prediction will still occur, but with less predictive accuracy.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default to 6 hours between checking projects for
advance
    // AND prediction calculations (e.g. completed/start when
    // conditions, due dates, etc.)
    bp.Vars.TimerSecondsCheckProjAdvance = 60 * 60 * 6;


    // default to 1 hour between checking projects for email
reminders
    bp.Vars.TimerSecondsCheckProjReminders = 60 * 60 * 1;

    // Less accurate user predictions for task completion,
    // but fewer system resources used
    bp.Vars.fDisableUserPrediction = true;
}
```

AllowedExportLocations

The AllowedExportLocations variable is a list of strings that contains all of the folder locations to where a file export via Knowledge View, Custom Task or other file location export is allowed.

For instance, this variable enables you to control the locations to which a KView can export documents when called via a URL. Calling a Knowledge View via URL will accept a URL Parameter named `exportname` that enables the Knowledge View to export a file to a specified file path/file name. Similarly, a file export from a Custom Task will usually provide a file path property to specify an export location for an exported document attachment.

To prevent users from exporting files to unwanted locations, or overwriting existing files, this variable MUST be set. Any attempt to export a file to any folder not listed in this variable will fail. The locations listed in this system variable will be treated as parent-level folders, which means that you can write documents to any location or subfolder *below* the specified folder location.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Create List values
    List<string> MyExportLocations = new List<string>();
    MyExportLocations.Add(@"C:\Documents");
    MyExportLocations.Add(@"C:\Files\Images");

    // Apply List values to the Custom Variable
    bp.Vars.AllowedExportLocations = MyExportLocations;
}
```

fEnableEncryptionMigration

This variable, when set to "false", will disable the migration to AES encryption when upgrading to Process Director v5.44.700 or higher from v5.44.600 or lower. The default value for this variable is "true".

> ⊕ **The encryption system used by Process Director prior to v.5.44.600 has been deprecated. Setting this value to "false" will prevent encrypted fields from being upgraded to full AES encryption, and existing encrypted fields will use the deprecated encryption system, which is being replaced by AES.**

When this value is set to "false", though migration of existing encrypted values won't occur on upgrade, **new encrypted values will still be created using AES**, e.g., submitting a new form instance with encrypted form controls.

Any time data that uses the deprecated encryption is accessed, an ERROR log message will be generated as a reminder that migration hasn't yet been performed (I.e., "ERROR: Deprecated encrypted value detected, please upgrade using Administration pages."). There won't be any data-loss or corruption, as the system will recognize both AES and deprecated encryption.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Turn off AES Encryption Migration
    bp.Vars.fEnableEncryptionMigration= false;
}
```

fAllowCustomUserString

This Custom Variable, when set to "true", enables you to call the custom functions called `UserDisplayString()` and `UserDisplayString2()` in the vars.cs.ascx Customization File. These functions enable you to control what the display string looks like for a user in the product.

The `UserDisplayString()` function sets the display string universally in the product.

The `UserDisplayString2()` function returns the Display string to the location/object from which it is being called, to give you the flexibility to only change the display string for the user in a specific place instead of in the whole product.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enables you to call the UserDisplayString and User-
DisplayString2 functions.
    bp.Vars.fAllowCustomUserString = true;
}
```

fAuthWindows

This variable, when set to true, enables the use of Windows login security. The default value for this variable is "false", and is set in the XSD file.

You usually don't turn on this feature on by changing the value of this System Variable. Instead, this feature is enabled in the product in the User Authentication settings by setting the Enable Windows Authentication dropdown to "True" .Windows login security can't be implemented until the Enable Windows Authentication setting is set to "True".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Windows authentication is disabled
    bp.Vars.fAuthWindows = false;
}
```

fAuthWindowsIntegrated

This variable, when set to true, enables the use of NTLM integration. The default value for this variable is "true", and is set in the XSD file.

While this variable is set to "true" by default, Windows Integrated Security is set to "false' by default. You usually turn on this feature in the product in the User Authentication settings by setting the Enable Windows Authentication dropdown to "True". Even though the fAuthWindowsIntegrated variable is set to true by default, windows login security can't be implemented until the Enable Windows Authentication setting is set to "true".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // NTLM Integration is disabled.
    bp.Vars.fAuthWindowsIntegrated = false;
}
```

fAutoDST

This variable enables you to override whether daylight savings is used in the time zone.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Use CST for everyone
    bp.Vars.sTimeZoneID = "Pacific Standard Time";

    // Do not use daylight savings time adjustments
    bp.Vars.fAutoDST = false;
}
```

fDisableExcelImport

This variable enables you to control whether the Excel files can be used to auto-matically populate SQL databases.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable the auto-excel import feature
    bp.Vars.fDisableExcelImport = false;
}
```

fDisableKViewAppCaches

This variable, when set to "true", will disable application level caching in the Know-ledge Views when using a Load Balanced system. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable application-level caching for load balancing
    bp.Vars.fDisableKViewAppCaches = true;
}
```

fDisable_sValueSearch

This variable, when set to "true", will disable the time-consuming update of the sValueSearch field that takes place during some upgrades. This option enables the system to be updated and work without using the sValueSearch field. Upgrades that require this field update can take up to 24 hours. The sValueSearch field was added to the database to significantly improve search performance. It must be pop-ulated, however, which can be a very time-consuming process. This population, by default, happens during the upgrade cycle, which can cause the upgrade to take a very long time.

You can, when this value is set to "true", skip the field update during the upgrade, and populate it later using the Troubleshooting section in the IT Admin area in DEBUG mode. There is a link to update the search value column in tblFormData. This command could take 24 hours to run, depending on the speed of your

database. It can be run and re-run and, each time, will pickup where it left off, until the entire update is complete. This feature enables you to perform the update during non-work hours. Once the update has no more records to process you can remove this variable setting from the vars file to have the searches start using the new sValueSearch column.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable the update to the sValueSearch field
    bp.Vars.fDisable_sValueSearch = true;
}
```

fEnableDenyPermissions

This variable enables Process Director to implement a "Deny" permissions model, so that users can be specifically denied access to objects. In most cases, the default Process Director permissions model is adequately hardened, so this variable is set to "false" by default. There are, however, some use cases for adding specific denial records. Setting this variable to "true" will turn on the denial permissions model, but the extra permissions checking that the denial model requires may have some impact on system performance.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will enable denial permissions.
    bp.Vars.fEnableDenyPermissions = true;
}
```

fEnableFormFieldDownload

This variable, when set to "true" enables users to download the form fields in an excel file, from a link on the Properties tab of the Form Definition. The default value is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Allow form field downloads in Excel.
    bp.Vars.fEnableFormFieldDownload = true;
}
```

fEnableJSURL

This variable, when set to "true", enables custom forms to pass JavaScript to be executed after a form completes. The default value for this variable is "false". For users of Process Director v4.54 and below, this variable must be set to "true" to enable the Copy Form Data Custom Task to close one form and open another, when configured to do so.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will enable custom forms to pass JavaScript to be
executed
    // after the form completes.
    Vars.fEnableJSURL = true;
}
```

fEnableOldShowAttach

This variable, when set to "true", disables the Group Name wildcard functionality for ShowAttach controls. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable wildcards for ShowAttach control Group Names
    bp.Vars.fEnableOldShowAttach = true;
}
```

fEnableReferrerProtection

This Boolean variable, when set to "true," enables Process Director to screen for Cross-Site Request Forgery (CSRF) by ensuring that the HTTP Referrer header is

valid. CSRF is a type of malicious exploit of a website where unauthorized com-mands are transmitted from a user that the website trusts. The default value for this variable is "false".

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // HTML header validation to protect against CSRF
    Vars.fEnableReferrerProtection = true;
}
```

fRemoveSavedInstForOldUsers

This variable, when set to "true", will remove all saved form instances for a user that is deleted or disabled. The default value for this variable is "false".

> ⊗ **Caution should be used when implementing this feature, as removing or disabling a user will delete all historical form data for the user.**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Remove form instances for deleted/disabled users
    bp.Vars.fRemoveSavedInstForOldUsers = true;
}
```

fSharedDelegationAllProcesses

This variable , when set to "true" will implement Shared Delegation for all pro-cesses. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will enable shared delegation for all processes.
    bp.Vars.fSharedDelegationAllProcesses = true;
}
```

ForceSecureCookies

This string variable enables you to set cookies to transmit using only the Secure Sockets Layer (SSL).

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Force cookies to transmit only via SSL.
    bp.Vars.ForceSecureCookies = true;
}
```

sMobileWebServerURL

This string variable enables customers who have licensed the offline Appenate server to configure the custom URL of their mobile server for use by Custom Tasks and other features. This feature is unavailable for versions prior to Process Director v5.45, and who have not also licensed the offline mobile feature.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Mobile App server URL
    bp.Vars.sMobileWebServerURL = "http://mobile.bplogix.com";
}
```

fTestMode

These variables allow you to set the system into a test mode. In this mode, Windows Integrated authentication is disabled. This mode enables anyone to log into the server without a password. Use this setting with caution. It should only be used on non-production systems to test processes and Forms.

For more information, see the Test Server Methodology topic in the System Administrator's Guide.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // **************************************
    // SET THE TEST USER FOR TEST MODE
    // Turn test mode on
    bp.Vars.fTestMode = true;

    // Set the test email user
    bp.Vars.TestUserEmails = User.GetUserByUserID(bp, "my_
test_id");

    // ===OR===

    // **************************************
    // SET THE TEST USER EMAIL ADDRESS FOR TEST MODE
    // Turn test mode on
    bp.Vars.fTestMode = true;

    // Set the test email user
    bp.Vars.TestUserEmailAddress = "username@domain.com";
}
```

fUseNewLoginSessionGUID

This variable, when set to "true", will cause the users session GUID to be cleared any time a new login occurs. This setting will automatically relog a user who signs in using same user ID, even if that login is still active on the same computer in a different browser window. Relogging the user will clear all existing session data, including session variables, and will initiate a new session for the user.

This variable is configured in the `PreSetSystemVars()` *function.*

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will clear an existing users active session if the
user logs in again
    bp.Vars.fUseNewLoginSessionGUID = true;
}
```

fWebServiceAuth

This boolean determines whether authorization is required for Web Service users. The default value is true.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Do not require web service authorization
    bp.Vars.fWebServiceAuth = false;
}
```

Locales

This string variable enables you to add additional cultures to process director. In most cases, each additional culture will have a resource file for localizing strings for different cultures, as described in the Localization topic of the Developer's Guide. Once the cultures have been added, they'll appear in the `Culture` property dropdown controls that appear in each user's profile, and in other places.

Parameters

- **pName:** The string name of the culture, e.g. "Spanish"
- **pValue:** The string value used to identify the culture, in one of the standard culture formats, e.g. "es". Process Director will use this value to identify the Culture name used on the name of the RESX file, e.g., "strings.es.resx".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Add Cultures to process Director
    bp.Vars.Locales.Add(new NameValue("Spanish", "es"));
    bp.Vars.Locales.Add(new NameValue("German", "de"));
}
```

nMaxActivityStarts

If a Process Timeline instance is determined to be in an endless loop condition, the problem activity will be placed into an error state. This detection logic to define an endless loop is controlled through the nMaxActivityStarts, nMaxActivityStartsInLastSecs, and nTimelineLoopCountStarts variables.

This variable sets how many times a step restarts in the number of seconds set in the nMaxActivityStartsInLastSecs variable before considering it in a loop. The default value is 100.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.nMaxActivityStarts = 100;
}
```

nMaxActivityStartsInLastSecs

If a Process Timeline instance is determined to be in an endless loop condition, the problem activity will be placed into an error state. This detection logic to define an endless loop is controlled through the nMaxActivityStarts, nMaxActivityStartsInLastSecs, and nTimelineLoopCountStarts variables.

This variable sets how many seconds to use in loop calculations performed by the nMaxActivityStarts variable. The default is 60.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.nMaxActivityStartsInLastSecs = 60;
}
```

nTimelineLoopCountStarts

If a Process Timeline instance is determined to be in an endless loop condition, the problem activity will be placed into an error state. This detection logic to define an endless loop is controlled through the nMaxActivityStarts, nMaxActivityStartsInLastSecs, and nTimelineLoopCountStarts variables.

This variable sets the number of activities that may be started in one pass of a process. The default is 100.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set max number of allowable loops to 50
    bp.Vars.nTimelineLoopCountStarts = 50;
}
```

sPDFInterfaceURL

This variable is used to set the optional interface URL to use for converting PDF files.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the interface URL to use for converting
files to PDF.
    bp.Vars.sPDFInterfaceURL = "http://localhost/";
}
```

sPickupDirectoryLocation

IIS in Windows 2008 R2 and higher requires a path to be configured to the mail pickup directory when using the local SMTP server to send email. This is set using the sPickupDirectoryLocation custom variable. For Process Director v5.31 and higher, Process Director defaults this pickup directory to the Windows default of C:\Inetpub\mailroot\Pickup\ so it only needs to be set when it is in a different location. Additionally, the setting of this variable won't force emails to use the local SMTP server when an SMTP Host is specified in the installation settings.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Change the mail pickup directory to a non-default loc-
ation.
    bp.Vars.sPickupDirectoryLocation = "C:\nonDe-
fault\directory";
}
```

sTimeZoneID

This variable enables you to set the default time zone for the server and for all users. A time zone configured in a user profile will override this variable for that user. This string should be set to the Time Zone ID specified by the .NET environment. Leave this variable null to use the system time zone.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Use CST for everyone
    bp.Vars.sTimeZoneID = "Central Standard Time";

    // Automatically adjust for daylight savings
    bp.Vars.fAutoDST = true;
}
```

The list below shows the possible time zone values as defined by .NET:

Morocco Standard Time	GMT Standard Time	Romance Standard Time
W. Europe Standard Time	Central Europe Standard Time	Namibia Standard Time
Central European Standard Time	W. Central Africa Standard Time	Middle East Standard Time
Jordan Standard Time	GTB Standard Time	South Africa Standard Time
Egypt Standard Time	Syria Standard Time	E. Europe Standard Time
FLE Standard Time	Israel Standard Time	Russian Standard Time
Arabic Standard Time	Arab Standard Time	Arabian Standard Time
E. Africa Standard Time	Iran Standard Time	Georgian Standard Time
Azerbaijan Standard Time	Mauritius Standard Time	Ekaterinburg Standard Time
Caucasus Standard Time	Afghanistan Standard Time	India Standard Time
Pakistan Standard Time	West Asia Standard Time	Central Asia Standard Time

Sri Lanka Standard Time	Nepal Standard Time	Myanmar Standard Time
Bangladesh Standard Time	N. Central Asia Standard Time	China Standard Time
SE Asia Standard Time	North Asia Standard Time	W. Australia Standard Time
North Asia East Standard Time	Singapore Standard Time	Tokyo Standard Time
Taipei Standard Time	Ulaanbaatar Standard Time	Cen. Australia Standard Time
Korea Standard Time	Yakutsk Standard Time	AUS Eastern Standard Time
AUS Central Standard Time	E. Australia Standard Time	Vladivostok Standard Time
West Pacific Standard Time	Tasmania Standard Time	New Zealand Standard Time
Magadan Standard Time	Central Pacific Standard Time	Tonga Standard Time
Fiji Standard Time	Kamchatka Standard Time	Mid- Atlantic Standard Time
Azores Standard Time	Cape Verde Standard Time	SA Eastern Standard Time
E. South America Standard Time	Argentina Standard Time	Newfoundland Standard Time
Greenland Standard Time	Montevideo Standard Time	Central Brazilian Standard Time
Paraguay Standard Time	Atlantic Standard Time	Venezuela Standard Time
SA Western Standard Time	Pacific SA Standard Time	US Eastern Standard Time
SA Pacific Standard Time	Eastern Standard Time	Central Standard Time (Mexico)
Central America Standard Time	Central Standard Time	Mountain Standard Time (Mexico)

Canada Central Standard Time	US Mountain Standard Time	Pacific Standard Time
Mountain Standard Time	Pacific Standard Time (Mexico)	Samoa Standard Time
Alaskan Standard Time	Hawaiian Standard Time	
Dateline Standard Time	Greenwich Standard Time	

sLocalIPs

This variable is used to set the optional list of local IP addresses on the server. This is used so that any browser request from one of the local IP addresses will be able to access all administration functions.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the following IP addresses as "local" IP
    // addresses for this multi-NIC server
    bp.Vars.sLocalIPs = "10.1.5.14,10.1.5.32";
}
```

TestModeIPs

This variable enables you specify the IP addresses that are allowed to connect to a Process Director Server that has been placed in Test mode, via the fTestMode custom variable. This is required to allow access to a server that is in test mode. If accessing the server locally, the IP address doesn't need to be listed.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // ****************************************
    // SET THE TEST USER FOR TEST MODE
    // Turn test mode on
    bp.Vars.fTestMode = true;

    // Set the IP Addresses allowed to access the server
    bp.Vars.TestModeIPs.Add("10.1.1.1");
}
```

TestUserEmailAddress

This variable enables you to route all process emails (task list emails, notifications, etc.) to a single email address. This system variable doesn't require a database connection. Use this setting with caution. It should only be used on non-production systems to test process and Forms. If your installation is configured to enable users to retrieve forgotten passwords, via the fAllowRetrievePassword custom variable, the TestUserEmailAddress won't receive password reset request emails.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // ***************************************
    // SET THE TEST USER FOR TEST MODE
    // Turn test mode on
    bp.Vars.fTestMode = true;

    // Set test email
    bp.Vars.TestUserEmailAddress = "demo@bplogix.com";
}
```

TestUserEmails

This variable enables you to route all process emails (task list emails, notifications, etc) to a single user. Use this setting with caution. It should only be used on non-production systems to test processes and Forms.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // ***************************************
    // SET THE TEST USER FOR TEST MODE
    // Turn test mode on
    bp.Vars.fTestMode = true;

    // Specify user to use for test emails
    bp.Vars.TestUserEmails = User.GetUserByUserID(bp, "my_
test_id");
}
```

Auditing Custom Variables

You can alter some of the auditing functions of Process Director by editing the custom variables listed in this section.

fAuditAnonAccesses

This variable enables Process Director to log various events in the audit logs that are specific to anonymous access to Process Director. In most cases, the Process Director installation won't be accessed by outside users with anonymous access, so this variable is set to "false" by default. Setting this variable to "true" will turn on the audit logging for these events.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will turn on audit logging for anonymous access.
    bp.Vars.fAuditAnonAccesses = true;
}
```

fAuditFormAPICalls

This variable, when set to false, prevents Process Director from auditing form field changes made by API calls.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Do not audit form field changes made by API calls
    bp.Vars.fAuditFormAPICalls = true;
}
```

fAuditFormViews

This variable determines whether Process Director will write an audit log for all form views. This can result in a tremendous amount of audit logging when the value is set to "True". The default for this variable is "False".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Do not store audit logs when a user views
    // a Process Director Page. Setting this value to true may
cause
    // excessive logging and impact system performance.
    bp.Vars.fAuditFormViews = true;
}
```

fAuditLogFileOnly

This variable, when set to the default value of "true", stores the Audit log file in the file system only, in the /App_Data/log_archive/ folder. When set to false, the audit log is also stored in the Process Director internal database in addition to the file system.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Allow Process Director to store the audit log in the
    // internal database, in addition to the log file.
    bp.Vars.fAuditLogFileOnly = true;
}
```

nAuditLogDays

This variable enables you to control the number of days that Process Director should retain its audit logs. Logs will be archived in the /App_Data/log_archive/ folder. This custom variable is available only with users who have the Compliance edition for on-site installations, or a cloud installation. The default value for this variable is "7".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // archives logs for ten days
    bp.Vars.nAuditLogDays= 10;
}
```

nMaxAdminAuditRows

This variable enables you to set the maximum number of audit rows to display. The default is 1000.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of audit rows.
    bp.Vars.nMaxAdminAuditRows = 1000;
}
```

sAuditFieldStyle

This string variable enables you to provide custom CSS styling for the appearance of audit fields.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //These are the default CSS styles for audit fields.
    bp.Vars.sAuditFieldStyle = "background-col-
or:#FFFF99!important;";
}
```

sSkipAuditFields

This string variable enables you to optionally list field names for fields you wish to skip when auditing by providing a comma-separated list of field names. This will also enable you to skip all fields in an array or section.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.sSkipAuditFields = "fieldname1, fieldName2";
}
```

Audit Logging Variables

For installations that use the Compliance Edition of Process Director, some audit logging items are sent to the Windows Event Log, in addition to the audit logging

that you can access inside Process Director. The logging events sent to the Windows Event Log fall into two categories, Informational and Error entries.

Informational Entries

The following audit events are sent as informational entries to the Windows Event Log:

- Login
- SyncStart
- PasswordChange
- ReInit
- DelegateOn
- DelegateOff
- ImpersonateOn
- ImpersonateOff
- UserDisabled
- UserEnabled
- AdminReplaceUser
- CreateUser
- DeleteUser

Error Entries

The following audit events are sent as error entries to the Windows Event Log:

- AuthenticateFailed
- UserLockedOut

Additional Audit log events can be added to the default events by using the EventLogInfoAudits or EventLogErrorAudits custom variable in the Custom Variables file.

EventLogInfoAudits

This variable enables you to add an event Enum type to the Windows Event Log as an Informational entry.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    Vars.EventLogInfoAudits.Add(AuditType.AuthenticateFailed);
}
```

EventLogErrorAudits

This variable enables you to add an event Enum type to the Windows Event Log as an Error entry.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    Vars.EventLogErrorAudits.Add(AuditType.UserDisabled);
}
```

Default Settings Custom Variables

The system defaults are already defined in Process Director, but you can add to or override the default settings by editing the vars.cs.aspx file in the "\Program Files\BP Logix\Process Director\website\custom\" directory created during the installation.

BusinessHolidays

This variable enables you to control which dates are considered Holidays. The standard US holidays are included in the default set of business holidays.

New Year's Day: January 1

Birthday of Martin Luther King Jr: January 20

President's Day: February 17

Memorial Day: Last Monday in May

Independence Day: July 4

Labor Day: First Monday in September

Columbus Day: October 12

Veterans' Day: November 11

Thanksgiving: Last Thursday in November

Christmas: Friday, December 25

Please note that many holidays fall on a weekend, necessitating Friday or Monday holiday dates that are different from the official dates. Dates are, of course, hard-coded as the actual holiday dates in the system. For Cloud customers, BP Logix updates the default holidays routinely. For on-premise customers, however, the holidays dates need to be completely replaced on at least a yearly basis, to ensure the correct dates are marked as holidays in the system. Updating the product will automatically update the current default holidays, and this is the preferred solution for keeping the default holidays up to date. Non-US customers will need to completely replace the default US holiday calendar with a local calendar in the customization file.

You can simply add holidays to the default calendar by using the BusinessHolidays.Add() method. Similarly, you can use the BusinessHolidays.Remove() method to remove a specific holiday from the default list of holidays.

You can also completely replace the default holiday calendar with a calendar of your own configuration by creating a new BusinessHolidays HashSet, then using the BusinessHolidays.Add method to add the new holidays you wish to include in your calendar. You'll need to use the System.Collections.Generic namespace to create the new HashSet by declaring it, or using it inline.

Example

Datetime declarations are set in the format: year, month, day, so declaring May 12, 2021 as a holiday date would be done via the syntax:

```
bp.Vars.BusinessHolidays.Add(new DateTime(2021, 5, 12).Date);
```

Sample Code Block

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
//==========Add Holidays to the Default Cal-
endar==============
    // Christmas Eve, 2020
    bp.Vars.BusinessHolidays.Add(new DateTime(2020, 12,
24).Date);
    //Boxing Day (Canada, UK), 2020
    bp.Vars.BusinessHolidays.Add(new DateTime(2020, 12,
26).Date);
//===========================================================-
=

//==========Create a New Holiday Cal-
endar=====================
    // Add a completely new Holiday calendar and overwrite the
default
    // US holidays. You must declare the namespace (i.e.
    // Using System.Collections.Generic)or use the namespace
inline
    // to create a new HashSet

    //Create the new HashSet to overwrite the default holidays
    bp.Vars.BusinessHolidays = new HashSet<DateTime>();
    //Add the new holidays
    //New Year's Day
    bp.Vars.BusinessHolidays.Add(new DateTime(2020, 1,
1).Date);
    //Canada Day
    bp.Vars.BusinessHolidays.Add(new DateTime(2020, 7,
1).Date);
    //Independence Day (US)
    bp.Vars.BusinessHolidays.Add(new DateTime(2020, 7,
4).Date);
//===========================================================-
=
}
```

BusinessHourStart

This variable enables you to control when business hours start, and is used in calculations of process due dates. The default value is "8" (8:00 AM).

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This organization's business day starts at 7am
    bp.Vars.BusinessHourStart = 7;
}
```

BusinessHourStop

This variable enables you to control when business hours end, and is used in calculations of process due dates. The default Value is "17" (5:00 PM).

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This organization's business day ends at 5pm
    bp.Vars.BusinessHourStop = 17;
}
```

CheckReminderBusinessHours

This variable enables you to control if process reminders should be sent ONLY during business hours. The default value for this variable is "true".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Only send reminders during business hours
    bp.Vars.CheckReminderBusinessHours = true;
}
```

DefaultHTMLEncode

This variable enables you to set HTML as the default encode type for Form system variables.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Set HTML as the default encode type for Form System vari-
ables.
    bp.Vars.DefaultHTMLEncode = true;
}
```

DefaultInviteEmail

This variable enables you to set your own .ascx file as the default invite email

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom ASCX file to use as invite email
    bp.Vars.DefaultInviteEmail = "~/cus-
tom/YourCustomInviteEmail.ascx";
}
```

DefaultPasswordEmail

This variable enables you to set your own .ascx file as the default password reminder email. When constructing this email message, you'll want to create a link to the correct password change page for the user. Creating this link will require the use of a system variable formatter, PWD_GUID, which is a special formatter for use with password changes. This formatter returns the Password Change GUID for the user. This value is returned from the guidPwdChange field from the tblUser table in the Process Director database. This field value is saved to the database when the user requests a password reset and it is reset as soon as they log in.

To construct the URL for the password change page in your custom email, you can have the URL dynamically constructed using the following syntax:

`{INTERFACE_ URL}user_ password_ reset.aspx?pwdguid= {EMAIL_ USER,-format=PWD_GUID}`

The Interface URL system variable will return the interface URL setting from your system to return the base URL for the Process Director server, followed by a slash. The Email User system variable, using the PWD_GUID formatter, will return the

Password Change GUID that was set for the user when they asked for a password reset.

With this syntax, the requesting user will receive an email with the correct link to the password reset page for their specific password reset request.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom ASCX file to use as password reminder email
    bp.Vars.DefaultPasswordEmail = "~/cus-
tom/YourCustomPasswordEmail.ascx";
}
```

DefaultTimelineEmail

This variable enables you to set your own .ascx file as the default Timeline email.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom ASCX file to use as Timeline email
    bp.Vars.DefaultTimelineEmail = "~/cus-
tom/YourCustomDefaultEmail.ascx";
}
```

DefaultWorkflowEmail

This variable enables you to set your own .ascx file as the default process email template.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
// Custom ASCX file to use as process email template
    bp.Vars.DefaultWorkflowEmail="~/cus-
tom/YourCustomDefaultEmail.ascx";
}
```

DisabledTabsDisabled

> ⛔ **This Custom Variable was deprecated in Process Director v5.45.100, and should no longer be used. Instead, appropriate conditions should be assigned to tabs via the UI. It remains in the product for legacy customers only.**

This variable, when set to "true", enables users to disable individual tabs in form definitions that use the TabStrip/TabContent controls. The default for this option is "False".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable Tab Control disabling
    bp.Vars.DisabledTabsDisabled = true;
}
```

ForceMobileAdvanced

This variable, when set to "true", will make every connection to Process Director a mobile device connection. Even Desktop connections will be presented as mobile connections. This can he helpful for preventing some mobile devices from trying to use the Desktop UI for Process Director.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Force mobile connection
    bp.Vars.ForceMobileAdvanced = true;
}
```

ForceHttpOnlyCookies

This variable, when set to "true", will force the the "HttpOnly" attribute be used on session cookies in the browser. The default value for this variable is "false".

> ⓘ **Enabling this setting will prevent the use of the BP Logix plugin for editing Word-based forms.**

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Force HttpOnly attribute for session cookies
    bp.Vars.ForceHttpOnlyCookies = true;
}
```

ForceSecureCookies

This variable, when set to "true", will force the "Secure" attribute be used on session cookies in the browser. The default value for this variable is "false".

> ⓘ **This setting is only valid for https sessions.**

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Force secure cookies for HTTPS
    bp.Vars.ForceSecureCookies = true;
}
```

RefreshParentWorkspaces

This variable, when set to the default value of "true", enables the home workspace to be refreshed after completing tasks from an email. To disable this feature, set the value to "false".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Turn off workspace refresh
    bp.Vars.RefreshParentWorkspaces = false;
}
```

RemoveSIDFromJS

By default, this variable is set to "true" which prevents the Windows Session ID from being transmitted via JavaScript. Some older versions of Internet Explorer,

however, require Session IDs to be passed on the URL for popup windows that are opened via JavaScript. So, if you encounter this scenario, set this variable to "false".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Pass the SID via JavaScript to popup windows
    bp.Vars.RemoveSIDFromJS = false;
}
```

LDAP Custom Variables

You can alter the way Process Director interoperates with LDAP, defines LDAP users, authenticates with LDAP, and other functions by using the custom variables in this section.

Properties

fAuthFastLDAP

This a boolean value that determines whether "fast" LDAP authentication will be used. The default is False.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fAuthFastLDAP = true;
}
```

fAuthLDAP

This a boolean value that determines whether LDAP authentication is enabled. The default is False.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fAuthLDAP = false;
}
```

fAuthLDAPEx

This a boolean value that determines whether LDAP authentication using the extended LDAP API is enabled. The default is False.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fAuthLDAPEx = false;
}
```

fAuthLDAPAutoAdd

This a boolean value that determines whether LDAP users will automatically be added as Process Director users after authentication. The default is False.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fAuthLDAPAutoAdd = false;
}
```

LDAP_DisplayName_Field

This a string value that contains the name of the LDAP field containing the GUID property of the user record. In Active Directory, this would be "displayName".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAP_DisplayName_Field = "Display Name";
}
```

LDAP_Email_Field

This a string value that contains the name of the LDAP field containing the user's email address. In Active Directory, this would be "mail".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAP_Email_Field = "mail";
}
```

LDAP_GUID_Field

This a string value that contains the name of the LDAP field containing the GUID property of the user record. In Active Directory, this would be "objectGUID".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAP_GUID_Field = "ObjectGUID";
}
```

LDAPEx_ReferralChasing

When set to the default value of ReferralChasingOption.All, this variable tells the LDAP server to forward the authentication to all other servers. This is used when the LDAP server containing the user records is for a different domain than the domain against which you wish to authenticate.

The possible options for referral chasing are:

- **All:** Chase referrals of either the subordinate or external type.
- **External:** Chase external referrals.
- **None:** Never chase the referred-to server. Setting this option prevents a client from contacting other servers in a referral process.
- **Subordinate:** Chase only subordinate referrals which are a subordinate naming context in a directory tree. The ADSI LDAP provider always turns off this flag for paged searches.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAPEx_ReferralChasing =
```

```
ReferralChasingOption.All;
}
```

LDAP_PageSize

This a integer value that enables you to set a custom page size for the LDAP records to be returned. The default value is 0. Be advised that setting this value may cause some LDAP servers to crash. The maximum page size recommended for Active Directory is 1000.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAP_PageSize = 0;
}
```

LDAP_URL

This a string value that contains the URL of the LDAP server.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAP_URL = "ldap://LdapServerName.com";
}
```

LDAP_UserID_Field

This a string value that contains the name of the LDAP field containing the User ID. In Active Directory, this would be "sAMAccountName".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.LDAP_UserID_Field = "sAMAccountName";
}
```

List Maximum Custom Variables

These system variables enable you to specify the maximum number of records that are returned by various system lists, such as the maximum number of Knowledge View roes, User/Group Picker rows, etc.

nKViewBuiltinMaxResults

This variable enables you to set the maximum entries that any KView can return. The default value for this variable is "1000".

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set maximum number of KView returns to 900 records
    bp.Vars.nKViewBuiltinMaxResults = 900;
}
```

nMaxAdminPermRows

This variable enables you to set the maximum number of administrative permissions rows to display. The default is 1000.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of admin permissions
rows.
    bp.Vars.nMaxAdminPermRows = 1000;
}
```

nMaxAdminRows

This variable enables you to set the maximum number of rows in the Admin section for any list. The default is 100.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of rows to 400 in any
list of
    // the admin section.
    bp.Vars.nMaxAdminRows = 100;
}
```

nMaxAdminRowsGroupFilter

On the User Admin page, there is a search box that will enable you to search for a specific user by typing in a text string. By default, Process Director will include Group names in that search. However, for a large number of users, searching the Group names as part of this search adds unacceptable overhead to the search function. If the number of users in the system exceeds this value, the search function on the User Admin page will stop using Group names as part of the search. The default value is 10000.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of rows before stop-
ping
    // the use of Group names as a search criteria in User
Admin.
    bp.Vars.nMaxAdminRows = 10000;
}
```

nMaxBusinessValueRows

This variable sets the maximum number of rows that will be returned by a Business Value. The default is 200.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Default is 200
    bp.Vars.nMaxBusinessValueRows = 100;
}
```

nMaxGroupDropdownRows

This variable enables you to set the maximum number of dropdown rows that appear in a Group Picker control that is displayed as a dropdown control.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of user dropdown rows.
    bp.Vars.nMaxGroupDropdownRows = 15;
}
```

nMaxProfileButtons

This variable enables you to set the maximum number of buttons an administrator can configure in a profile. These buttons are show at the top of each user's home page, and the default is 12.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of buttons an admin can
    // configure.
    bp.Vars.nMaxProfileButtons = 12;
}
```

MaxUploadSize

This string variable enables you to set the maximum size, in bytes, for file uploads and attachments.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set max upload size to 1MB.
    CV.MaxUploadSize = 1000000;
}
```

nMaxUserDropdownRows

This variable enables you to set the maximum number of dropdown rows that appear in a User Picker control that is displayed as a dropdown control.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of user dropdown rows.
    bp.Vars.nMaxUserDropdownRows = 15;
}
```

nMaxUserPermsRows

This variable enables you to set the maximum number of user permissions rows to display. The default is 1000.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum number of user permissions
rows.
    bp.Vars.nMaxUserPermsRows = 1000;
}
```

Logs Custom Variables

These Custom Variables enable you to specify settings to control how Process Director's logging features operate.

fEnableDatabaseLogs

Process Director v4.55 and higher logs XML import events, goal evaluations, and Active Directory syncs by default, which means that the default value for this variable is "true". This logging can be turned off by setting this variable to false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Turn off database logging for imports, AD Syncs, etc.
    bp.Vars.fEnableDatabaseLogs = false;
}
```

fKeepADSyncInfoLogs

This variable, when set to "false", will prevent the [Active Directory Sync Log History](#) from from logging most events at the "info" level (i.e., it will only log warnings and errors). This should substantially reduce the number of logs in the tblLogs table in the database, but you may still want to clear out the old logs prior to setting this variable. You can do so by adjusting the [nImportLogSyncDays](#) variable to 0, then running the global timer routines in the Troubleshooting window. Once complete, you can reset [nImportLogSyncDays](#) to the desired value. This Variable can be used in conjunction with the [nMaxADSyncLogEvents](#) variable to control the length of AD synchronization audit logs. The default value for this variable is "true".

> ⓘ **Prior to Process Director v5.3, clearing out the old logs requires resetting the nImportLogDays Custom Variable. The nImportLogDays was deprecated in v5.3 and replaced with nImportLogSyncDays.**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent detailed AD Sync logging
    Vars.fKeepADSyncInfoLogs = false;
}
```

nArchiveLogDays

This variable enables you to control the number of days that Process Director should retain its logs. Logs will be archived in the /App_Data/log_archive/ folder. This custom variable is available only with users who have the Compliance edition for on-site installations, or a cloud installation.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.nArchiveLogDays = 7; // archives logs for a week
}
```

nImportLogDays

> ⊘ **This Custom Variable has been deprecated in Process Director v5.3.**

Process Director v4.55 and higher logs XML import events, goal evaluations, and Active Directory syncs by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep logs for 45 days
    bp.Vars.nImportLogDays = 45;
}
```

nImportLogGoalDays

Process Director v5.3 and higher logs goal evaluations by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep Goal logs for 2 days
    bp.Vars.nImportLogGoalDays = 2;
}
```

nImportLogImportDays

Process Director v5.3 and higher logs XML imports by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep XML import logs for 15 days
    bp.Vars.nImportLogImportDays = 15;
}
```

nImportLogKVRunDays

Process Director v5.3 and higher logs Knowledge View executions by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep KV run logs for 5 days
    bp.Vars.nImportLogKVRunDays = 5;
}
```

nImportLogMLPublishDays

Process Director v5.3 and higher logs Machine Learning object publications by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep ML Pub logs for 5 days
    bp.Vars.nImportLogMLPublishDays = 5;
}
```

nImportLogSArunDays

Process Director v5.3 and higher logs Stream Action runs by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep SA run logs for 5 days
    bp.Vars.nImportLogSArunDays = 5;
}
```

nImportLogSyncDays

Process Director v5.3 and higher logs Active Directory synchronizations by default, and stores the logs for 30 days. This variable enables you to change the duration from 30 days, to a number of days of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep AD Sync logs for 5 days
    bp.Vars.nImportLogSyncDays = 5;
}
```

nMaxADSyncLogEvents

This variable sets the maximum number of synchronization log events that will be displayed on the Active Directory Sync Log History page.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Default is 200
    bp.Vars.nMaxADSyncLogEvents = 100;
}
```

nMaxLogBackups

This variable enables you to set the maximum number of log files. After a log file reaches the size configured by nMaxLogFileSize, it will "wrap" in a circular log file queue.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Keep a maximum of 20 wrapping log files
    bp.Vars.nMaxLogBackups = 20;
}
```

nMaxLogFileSize

This variable enables you to set the maximum size of the wrapping log files. Specify this value in kilobytes.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum log file size to 3 meg.
    bp.Vars.nMaxLogFileSize = 3000;
}
```

Miscellaneous Variables

There are a variety of miscellaneous custom variables that you can set to determine how some Process Director features or functions work.

fAllowV6Import

This variable enables you to control whether the iMarkup Server V6 import option appears in the Content List.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fAllowV6Import = true;
}
```

fCONTAINSUseValueSearchOnly

When using full-text search operators, such as (Contains (Lexical) search operations may take a long time, especially when searching many records containing long text fields. This custom variable can speed up the operation of full text

searches by limiting the search to the first 256 characters of the field being searched. While this may substantially speed up the search operation, it also reduces the accuracy of the search results, as search terms that aren't included in the first 256 characters of the field will be ignored.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Speed up full-text searching
    bp.Vars.fCONTAINSUseValueSearchOnly = true;
}
```

fEnableMultFormFieldsInCols

This Boolean variable enables Knowledge View designers to mix the use of the form field chooser from the `Choose System Variable` dialog box and FORM system variables in Knowledge View columns in Knowledge Views that return results from multiple Form Definitions. The default value for this variable is "True". Setting this variable to "False" will require that all fields be chosen from the Field Chooser.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Allows mixing methods for choosing form fields in KView
Columns.
    bp.Vars.fEnableMultFormFieldsInCols = true;
}
```

fEnableTransOnKVIEW

When true, this option enables running a SQL transaction prior to running the command to run a Knowledge View. Once the Knowledge View has been run, the transaction is rolled back. Running the transaction will aide in database concurrency for complex filtering on the Knowledge View, using the IsolationLevel.ReadUncommitted functionality on SQL Server only. This isn't enabled for Oracle because Oracle only supports ReadCommitted and Serializable, which would commit the transaction to the database. The default value for this variable is "true". Setting the value to "false" will prevent the SQL transaction from running.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disables running the automatic SQL Transaction
    bp.Vars.fEnableTransOnKVIEW = false;
}
```

fEnableTransOnSELECT

Knowledge Views use a transaction that will aide in database concurrency for complex filtering on the Knowledge View. This is done by having a transaction around the KVIEW SELECT. The default is to enable this functionality, but it can be turned off by setting this in the custom vars:

This function is only applicable on SQL Server (not Oracle).

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disables running the automatic SQL Transaction
    bp.Vars.fEnableTransOnSELECT = false;
}
```

fFormDataTrans

When true, this system variable enables database transactions for certain form field updates.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enables transactions for some form field updates
    bp.Vars.fFormDataTrans = true;
}
```

fFormSaveUpdatesOnly

When true, this system variable ensures that, when saving (submitting) Forms, only modified fields will be updated in the database. This option's default setting is "true". This setting enhances system performance.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fFormSaveUpdatesOnly = true; // Saves only updated
form data
}
```

fFormSkipDisableFieldsSave

When set to "true", this system variable ensures that disabled form fields can't be edited via client JavaScript, and only set via server directives, when additional security is desired. For Cloud installations of Process Director, the default value for this variable has been set to "true". For all other versions, the default value of this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disables JavaScript changes to disabled fields
    bp.Vars.fFormSkipDisableFieldsSave = true;
}
```

fFormSkipHiddenFieldsSave

When set to "true", this variable ensures that hidden form fields can't be edited via client JavaScript, and only set via server directives, when additional security is desired. For Cloud installations of Process Director, the default value for this variable has been set to "true". For all other versions, the default value of this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disables JavaScript changes to hidden fields
    bp.Vars.fFormSkipHiddenFieldsSave = true;
}
```

fHideLabelsFromConditions

For Process Director v5.33 and higher, setting this variable to "true" hides certain controls (Comment Log, Buttons, Button Area, Embedded Sections, Routing Slip, Labels) from the condition builder. These fields are generally not used to store data, so may not be needed in the Condition Builder. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Hides non-data fields from the Condition Builder
    bp.Vars.fHideLabelsFromConditions = true;
}
```

fListenToEmailSetting

This variable when set to true, will force the activity/step to use the "participants when activity starts" checkbox to determine if an email should be sent. Previously an email was unconditionally sent when a user was added/reassigned in a running step/activity by an administrator. The default value for this variable is false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Use the task's email setting for administrative task
change notifications
    bp.Vars.fListenToEmailSetting = true;
}
```

fPDFCreateOtherAsAttachments

This variable enables you to control whether the CreatePDFFromDocument API will create a container PDF file for documents that can't be converted directly to a PDF file.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Create container PDF file for non-convertible documents
    bp.Vars.fPDFCreateOtherAsAttachments = true;
}
```

FormFieldsAllowDisabledURLUpdate

This method is configured in the PreSetSystemVars method of the Custom.Vars file. It enables you to specify the field names for fields whose values you wish to set using a URL parameter. Any field name specified by this method can have the value modified by a URL parameter on any Form that uses a field with the specified name.

For instructions on implementing this method, see the Form Definition URL topic in the Implementer's Reference.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Fields to set via URL Parameter
    bp.Vars.FormFieldsAllowDisabledURLUpdate.Add
("FieldName1");
    bp.Vars.FormFieldsAllowDisabledURLUpdate.Add
("FieldName2");
}
```

fReAuthFillUserID

When set to true, a Process Director user will have to reenter his User ID when prompted for re-authentication. This variable is set to false by default, which means that the User ID is filled in automatically on a Form's Re-Authenticate control.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Force reauthentication
```

```
    bp.Vars.fReAuthFillUserID = true;
}
```

fSkipNextPageCheck

When a user tries to view a page in Process Director while not logged in, he will be redirected to a login page. After he logs in, he will be directed back to the original page he was attempting to view. This variable is set to false by default. If it is set to false, then the user won't be redirected after logging in if the URL interface parameter is invalid. This flag may need to be set to true if the users are navigating through a firewall that changes the URL.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Redirect users after login
    bp.Vars.fSkipNextPageCheck = true;
}
```

fSkipWhereUsedCheck

This variable prevents all usage checking upon the deletion of an object on the Content List. When set to true, Process Director won't check for the locations in which an object is used. This variable need only be set to true if Process Director takes too long to present a user with a dialogue showing him where the object is used upon deletion.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Do not check for object locations
    bp.Vars.fSkipWhereUsedCheck = true;
}
```

fUseAsyncUpload

By default, file uploads for Form attachments occur synchronously with Form submissions, meaning that the Form submission can't complete until all files are uploaded. Setting this value to "true" enables the form to be submitted while a

separate process performs the file uploads . This can prevent Form submissions from hanging as they wait for large file uploads, or low bandwidth.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enables files uploads to be processed separately
    // from Form submissions
    bp.Vars.fUseAsyncUpload = true;
}
```

fWebServiceAllowCredentialsURL

This variable enables callers of web services using REST to pass authentication credentials on the URL. The default value of this variable is "true". If this variable is set to "false", the fWebServiceAuth custom variable will also have to be set to false, as there will be no way to pass credentials for authorization.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable Authentication credentials
    // Requires also setting the fWebServiceAuth var to false
    bp.Vars.fWebServiceAllowCredentialsURL = true;
}
```

nDBCommandTimeout

This variable enables you to set the default database timeout value (normally 30 seconds).

For Process Director v6.1.200 and higher, the value set here will also govern the database timeout values for Report objects that use SQL commands to return data.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set the DB timeout to 60 seconds
    bp.Vars.nDBCommandTimeout = 60;
}
```

nDBTransIsolationLevel

This variable enables you to override the default database transaction isolation level if set to any value other than "0" or "-1". The default is "0".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Override the default database transaction isolation
level
    bp.Vars.nDBTransIsolationLevel = 1;
}
```

nDebugProcessTimeFactor

This system variable enables you to multiply the configured lengths of Timeline activities by a given factor. This can be used for test purposes, to shorten the duration of timeline activities relative to their configured due dates. Set this system variable to the factor by which you wish to multiply the configured activity durations.

This only works for due dates configured as relative times (e.g. 5 business days). It has no effect on setting due dates to a form field or a system variable. Use of this feature will also trigger the internal timer processing and activity checks on all pages which may hinder performance when enabled.

Example

```
public override void PreSetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Activities will take 10% as long as they are
    // configured to take
    bp.Vars.nDebugProcessTimeFactor = 0.1;
}
```

nLimitSearchToChars

This variable limits the number of characters to be searched in a form field. This optional setting only applies when searching for form data using "contains" form data in the filter. The default is no limit. This variable can provide a performance improvement when searching for form data on systems with millions of rows.

Example

```
public override void PreSetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Limit the search to only the first 256 characters in
the form data fields.
    Vars.nLimitSearchToChars = 256;
}
```

nWSTimeout

This system variable specifies the time, in milliseconds, that a web service call should wait before timing out.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // The default timeout time is 10 minutes
    bp.Vars.nWSTimeout = 1000 * 60 * 10;
}
```

sMobileAdvancedTypes

This variable will enable process director to detect additional browser strings for mobile devices.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will add additional browser agent strings for
mobile devices
    bp.Vars.sMobileAdvancedTypes = "Android 10, iOS 13, [Some
Other Browser Type]";
}
```

sUploadAddCookie

This variable enables you to identify an HTTP cookie that should be added to the document and Form upload web service call. Some session managers require a cookie for all page access. If this variable is specified, the named cookie will be copied to the upload requests using obj_upload.aspx.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SiteMinder requires the SMSESSION cookie
    bp.Vars.sUploadAddCookie = "SMSESSION";
}
```

ValidationPhonePattern

This variable contains a regular expression used to validate form fields containing phone numbers. See the example for the default value of this variable:

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Regular expression to use for validating phone numbers
    bp.Vars.ValidationPhonePattern =
     @"^(?:(?:[\+]?(?<CountryCode>[\d]{1,3}(?:[ ]+|[\-
     .])))?[(]?(?<AreaCode>[\d]{3})[\-/)]?(?:[ ]+)?)?
     (?<Number>[a-zA-Z2-9][a-zA-Z0-9 \-.]{6,})(?:(?:[
     ]+|[xX]|(i:ext[\.]?)){1,2}(?<Ext>[\d]{1,5}))?$";
}
```

ValidationUrlPattern

This variable contains a regular expression used to validate URL strings. See the example for the default value of this variable:

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Regular expression to use for validating URLs
    bp.Vars.ValidationUrlPattern = @"^(http|https|ftp)\://
     [a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(:[a-zA-Z0-9]*)
     ?/?([a-zA-Z0-9\-\._\?\,\'/\\\+&amp;%\$#\=~])*$";
}
```

ValidationZipCodePattern

This variable contains a regular expression used to validate form fields containing ZIP codes. See the example for the default value of this variable:

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Regular expression to use for validating zip codes
    bp.Vars.ValidationZipCodePattern = @"^(\d{5}-\d{4}|\d{5}\d
{9})$";
}
```

ML and AI Custom Variables

Process Director v5.0 and higher supports custom variables for Machine Learning (ML) and Artificial Intelligence (AI) services. Some of the services are provided by third parties, and require external accounts to access those services, while others are integral to Process Director.

Google Sentiment

The Google sentiment service can analyze text submissions for the sentiment the text expresses. Passing a text string to the Google Sentiment service submits it for analysis, and the Service returns a double-precision number that ranges from -1.00 to 1.00. A value of 0 is a neutral sentiment. A negative value indicates negative sentiment, while a positive value indicates a positive sentiment. The closer to -1.00 or 1.00 the return value is, the more intense the sentiment expressed. For instance, a return value of 0.85 is a very positive sentiment.

Required Variables

To configure Process Director for the Google Sentiment service, the custom variables below must be configured:

GoogleSentiment_private_key

This string variable contains the your private key for the Google Sentiment service. This value is provided to you by Google.

GoogleSentiment_client_email

This string variable contains the email address you used as the client email address for the Google Sentiment service.

Optional Variables

GoogleSentiment_project_id

This string variable contains the Project ID for the Google Sentiment service. This value is provided to you by Google.

GoogleSentiment_private_key_id

This string variable contains the Private Key ID for the Google Sentiment service. This value is provided to you by Google.

GoogleSentiment_client_id

This string variable contains the Client ID for the Google Sentiment service. This value is provided to you by Google.

GoogleSentiment_client_x509_cert_url

This string variable contains the URL you used as the x509 Client Certificate for the Google Sentiment service.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //The following vars are required:
    bp.Vars.GoogleSentiment_private_key = @"YourPrivateKey";
    bp.Vars.GoogleSentiment_client_email =
"email@address.com";

    //The following vars are optional:
    bp.Vars.GoogleSentiment_project_id = "YourProjectID";
    bp.Vars.GoogleSentiment_private_key_id =
"YourPrivateKeyID";
    bp.Vars.GoogleSentiment_client_id = "YourClientID";
    bp.Vars.GoogleSentiment_client_x509_cert_url =
        "https://YourClientCertURL.com";
}
```

Mobile Application Custom Variables

Process Director v5.45 and higher enables a separately-licensed Mobile Application Component. This component requires custom variables to be configured to implement the component.

The custom variables will be configured by BP Logix for Cloud customers.

nAppenateCompanyID

This numeric variable specifies the Company ID for the mobile application of this Process Director installation.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Mobile Server ID
    bp.Vars.nAppenateCompanyID = 1;
}
```

fEnableFormFieldDownload

This boolean variable, when set to true, enables the display of the Download Field List action link on the Properties tab of Form definitions.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable Download Field List
    bp.Vars.fEnableFormFieldDownload = true;
}
```

sAppenateIntegrationKey

This string variable specifies the Mobile Server integration Key for the Process Director installation.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Mobile Server Integration Key
    bp.Vars.sAppenateIntegrationKey = "XXXXXXXX-XXXXXXXXX";
}
```

sMobileWebServerURL

This string variable specifies the URL of the mobile server to use for this Process Director installation.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Mobile Server URL
    bp.Vars.sMobileWebServerURL =
"https://mobile.bplogix.com";
}
```

Password Enforcement Custom Variables

Password enforcement settings variables are only for users using the Compliance or Cloud versions, and are set in the custom vars file to enforce password strength/security.

ForcePwdChangeEvery

This is an integer value that the number of days to elapse before requiring users to change their passwords.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Number of days before password change is required
    bp.Vars.ForcePwdChangeEvery = 30;
}
```

ForgotPasswordRedirectURL

This variable enables you to set the redirect URL to which the user will be referred when the "I forgot my password" link is selected.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Redirect for forgatten password
    bp.Vars.ForgotPasswordRedirectURL = "https://-
some.url/somePage.htm";
}
```

fUnlockAcctOnPasswordReset

This variable, when set to true, will automatically unlock a locked user account when they perform a password reset. The default value for this variable is false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Unlock account on user password reset
    bp.Vars.fUnlockAcctOnPasswordReset = true;
}
```

LoginFailuresUntilLock

This is an integer value that sets the number of allowed login failures until the account is locked.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Number of allowed login attempts
    bp.Vars.LoginFailuresUntilLock = 3;
}
```

NotifyPwdChangeDays

This variable enables you to set the number of days prior to password expiration to notify the user that a built-in account password is set to expire.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Notify users 3 days before their password expires
    bp.Vars.NotifyPwdChangeDays = 3;
}
```

PasswordResetRedirectURL

This variable enables you to set the redirect URL to which the user will be referred after a password reset.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Redirect users to this page after a password reset
    bp.Vars.PasswordResetRedirectURL =
"https://www.SomeURL.Com/SomePage.htm";
}
```

PwdMinLength

This is an integer value that sets the minimum length of the Password.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Minimum number of password characters
    bp.Vars.PwdMinLength = 10;
}
```

PwdMinLetters

This is an integer value that sets the minimum number of letter characters required.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Minimum number of password letter characters required
    bp.Vars.PwdMinLetters = 1;
}
```

PwdMinLower

This is an integer value that sets the minimum number of lower case characters required.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Minimum number of password lower-case letter characters
required
    bp.Vars.PwdMinLower = 1;
}
```

PwdMinUpper

This is an integer value that sets the minimum number of upper case characters required.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Minimum number of password upper-case letter characters
required
    bp.Vars.PwdMinUpper = 10;
}
```

PwdMinNumbers

This is an integer value that sets the minimum number of numeric characters required.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Minimum number of password numeric characters required
    bp.Vars.PwdMinNumbers = 1;
}
```

PwdMinSymbols

This is an integer value that sets the minimum number of symbol or special characters required. Any special character is allowed.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Minimum number of password special characters required
    bp.Vars.PwdMinSymbols= 1;
}
```

PwdNoReuseDays

This is an integer value that sets the minimum number of days that must elapse before a password can be reused.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // can't reuse a password in this many days
    bp.Vars.PwdNoReuseDays = 365;
}
```

PwdNoReuseNumTimes

This is an integer value that sets the minimum number of password changes that must elapse before a password can be reused.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // can't reuse a password in this many changes
    bp.Vars.PwdNoReuseNumTimes = 10;
}
```

PwdStrength

This variable enables you set a password strength for user passwords. The default value is "PasswordStrength.Low" which doesn't implement any password enforcement for user accounts. You can set the following custom password strengths:

`PasswordStrength.Low`: Requires that the password be at least 4 characters in length.

`PasswordStrength.Medium`: Requires that the password be at least 8 characters in length, and contain at least one letter and one number.

`PasswordStrength.High`: Requires that the password be at least 10 characters in length, and contain at least one number, one upper case letter, one lower case letter, and one symbol character.

`PasswordStrength.Custom`: Enables you to set a custom password strength using the variables discussed in this topic.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set the required password strength
    bp.Vars.PwdStrength = PasswordStrength.High;
}
```

PwdStrengthMessage

This variable enables you set a string displayed to the users when they change their password (for built-in users only). This string can be used, for example, to inform users of password strength requirements.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set a message to display to users changing
their
    // passwords
    bp.Vars.PwdStrengthMessage = "Passwords must be a minimum
of 8
      characters and have at least one number";
}
```

Setting Custom Password Enforcement Variables

To set a custom password strength in the custom variables file (vars.cs), you must first set the Password Strength to custom, then add the specific password strength variables that you desire. Below is some sample code for setting a custom password strength.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set a custom password strength
    bp.Vars.PwdStrength = PasswordStrength.Custom;
    bp.Vars.ForcePwdChangeEvery = 90;
    bp.Vars.LoginFailuresUntilLock = 5;
    bp.Vars.PwdMinLength = 8;
    bp.Vars.PwdMinLower = 1;
    bp.Vars.PwdMinUpper = 1;
    bp.Vars.PwdMinNumbers = 1;
    bp.Vars.PwdMinSymbols= 1;
    bp.Vars.PwdStrengthMessage = "Password must be at least 8
characters,
      and must have an upper case letter, a lower case letter,
a
      number, and a special character. EXAMPLE: m0squiTo!";
}
```

Process Administration Custom Variables

You have the option to alter who may use some Process Director functions by restricting their use to Process Administrators. The custom variables in this section define which actions can be restricted to process administrators.

fDisableAsyncWorkflow

This boolean variable, when set to true, will disable any asynchronous process steps in a process definition. The default value is false. This setting is primarily for use in development systems.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fDisableAsyncWorkflow = true; // Disable async pro-
cesses
}
```

fInternalDSPadminOnly

This boolean variable sets whether only Process Administrators are allowed to con-figure internal Datasources. The default value is true.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fInternalDSPadminOnly = true; // Only P-Admins can
config
}
```

fInternalUserDSPadminOnly

This boolean variable sets whether only Ptocess Administrators are allowed ro configure internal user Datasources. The default value is true.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fInternalUserDSPadminOnly = true; // Only P-Admins
can config
}
```

fNewSkipPendingLogic

For Process Director v5.36 and higher, the logic for evaluating the Activity Result system variable was changed to skip activity instances that were marked as not needed. This can cause an issue for customers that used the system variable that on conditions to control looping **when using the legacy Branch Activity Type**, instead of the Looping conditions on a Parent Activity Type. Setting this variable to "false" disables the new logic. The default value for this variable is "true".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Use the old Activity Result logic for Branch activities
    bp.Vars.fNewSkipPendingLogic = false;
}
```

fReportViewsPadminOnly

This boolean variable sets whether only Process Administrators are allowed to configure views in reports. The default value is true.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fReportViewsPadminOnly = true; // Only P-Admins
can config
}
```

fScriptsPadminOnly

This boolean variable sets whether only System Administrators are allowed to con-figure scripts in Forms, Process Timelines, and Knowledge Views. The default value is false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.fScriptsPadminOnly = false; // Anyone can config
}
```

nAsyncSubProcessWaitSecs

This variable enables you to control the number of seconds that Process Director should wait for an asynchronous subprocess to complete. On some systems, when starting a subprocess using the "Run Asynchronously" and the "Wait for sub-process to complete" options, the system can mark the calling task as complete before the called subprocess completes. This may be especially true if the sub-process contains complex rendering operations. This variable adds a 5-second wait time as a default, to help ensure the subprocess has time to complete.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Wait 5 seconds before marking the
    // calling task as complete
    bp.Vars.nAsyncSubProcessWaitSecs = 5;
}
```

Reporting Tool Custom Variables

These variables are available in Cloud Installations, or on-premise installations with the Advanced Reporting option. The custom variables in this section enable you to change the default settings of the reporting tool included with the advanced reporting option.

fReportShowExportTo...

The reporting tool has several Boolean variables that determine whether reports can be exported to specific file formats. All of these variables have similar names, and are all listed below, along with their default values.

Export Variables

VARIABLE NAME	EXPORT FILE FORMAT	DEFAULT VALUE
fReportShowExportToBmp	Bitmap Image	False
fReportShowExportToCsv	Comma-separated Text	True
fReportShowExportToDbf	dBase	False
fReportShowExportToDif	Data Interchange Format	False
fReportShowExportToDocument	Word 2010 and higher	False
fReportShowExportToExcel	Excel 2010 and higher	False
fReportShowExportToExcel2007	Excel 2007	True
fReportShowExportToExcelXml	Excel XML	False
fReportShowExportToGif	GIF Image	False
fReportShowExportToHtml	HTML	True
fReportShowExportToJpeg	JPEG Image	False
fReportShowExportToMetafile	Enhanced Metafile	False
fReportShowExportToMht	MHTML Archive	True
fReportShowExportToOpenDocumentCalc	Open Document Format for LibreOffice Calc	False

VARIABLE NAME	EXPORT FILE FORMAT	DEFAULT VALUE
fReportShowExportToOpenDocumentWriter	Open Document Format for LibreOffice Writer	False
fReportShowExportToPcx	PCX Image	False
fReportShowExportToPdf	Adobe PDF	True
fReportShowExportToPng	PNG Image	True
fReportShowExportToPpt	PowerPoint	True
fReportShowExportToRtf	Rich text File	True
fReportShowExportToSvg	Scalable Vector Graphics	False
fReportShowExportToSvgz	Compressed Scalable Vector graphics	False
fReportShowExportToSylk	Symbolic Link (Microsoft)	False
fReportShowExportToText	Text	True
fReportShowExportToTiff	TIFF Image	True
fReportShowExportToWord2007	Word 2007	True
fReportShowExportToXml	XML	True
fReportShowExportToXps	Open XML	False

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Do not show export to BMP option
    bp.Vars.fReportShowExportToBmp = false;
}
```

BaseURLFromRenderingServer

This string variable contains the base URL for the production server for which rendering operations will be performed, and should be configured only on a licensed

Rendering Server.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Base Server URL
    bp.Vars.BaseURLFromRenderingServer = "http://-
productionserver.com";
}
```

ReportRemoteURL

This string variable contains the URL for a licensed Rendering Server, and should be configured only on the production server.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Rendering Server URL
    bp.Vars.ReportRemoteURL = "http://renderingserver.com";
}
```

sReportInterfaceURL

This string variable contains the URL for the report viewer interface, if you wish to exercise the option to use a different URL for the report interface.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Report Server URL
    bp.Vars.sReportInterfaceURL = "http://server-
name.com/reports.aspx";
}
```

REST Custom Variables

Custom variables in this section of the documentation can be used to customize a variety of settings associated with using a Business Value to return data from a REST data source.

DefaultBVRestAccept

A string variable that determines the default format for results from a REST web service call by a Business Value. The default value is "XML".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set default REST returns as XML
    bp.Vars.DefaultBVRestAccept = "XML";
}
```

DefaultBVRestMethod

A string variable that determines the default method for returning results from a REST web service call by a Business Value. The default value is "POST".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set default REST return method
    bp.Vars.DefaultBVRestMethod = "POST";
}
```

DefaultBVRestContentType

A string variable that determines the default REST Content Type for a web service call by a Business Value. The default value is "application/XML".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set default REST content type
    bp.Vars.DefaultBVRestContentType = "application/XML";
}
```

DefaultBVRestHeaders

This Custom Variable implements an Add method that enables you to specify custom REST HTTP headers for a web service call by a Business Value.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Add REST header
    bp.Vars.DefaultBVRestHeaders.Add(new NameValue("Name",
"Value"));
}
```

DefaultBVRestCredentials

This Custom Variable enables you to specify the default network credentials to supply to a REST web service call by a Business Value.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default REST credentials
    bp.Vars.DefaultBVRestCredentials =
      new System.Net.NetworkCredential("UserID","password");
}
```

SAML Custom Variables

Custom variables in this section of the documentation can be used to customize a variety of settings associated with Using SAML/Federated Identity when working with SAML providers.

AddSAMLGroups

A Boolean variable that determines whether Process Director should add any Groups from a SAML login that Process Director doesn't already have.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Automatically create new SAML groups
    bp.Vars.AddSAMLGroups = true;
}
```

AddSAMLGroupsIgnore

This variable consists of a list of Group names to ignore in the auto-add logic (AddSAMLGroups)on the SAML login. E.g. if the SAML login includes an "admin" group, but you don't want a SAML login to automatically add anyone to the pre-existing "admin" group in Process Director.

Examples

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SAML Groups to ignore in the SAML import
    List<string> IgnoreGroups = new List<string>();
    IgnoreGroups.Add(@"admin");
    IgnoreGroups.Add(@"Administrators");
    bp.Vars.AddSAMLGroupsIgnore= IgnoreGroups;
}
```

OR

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SAML Groups to ignore in the SAML import
    bp.Vars.AddSAMLGroupsIgnore = new List<string> { "admin",
"Administrators" };
}
```

EXT_User_AutoCreate

A Boolean variable that determines whether Process Director should automatically create user accounts for externally authenticated users. The default value is "true". Setting the variable to "false" will prevent the automatic creation of user accounts.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Automatically create user accounts for externally
authenticated users
    bp.Vars.EXT_User_AutoCreate = true;
}
```

EXT_User_AutoCreateDisabled

A Boolean variable that determines whether the user accounts that are auto-matically created from external authentication should be initially set as disabled. The default value is "false". Setting the variable to "true" will initially set auto-matically created user accounts as disabled.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Initially set automatically created user accounts as
disabled
    bp.Vars.EXT_User_AutoCreateDisabled = true;
}
```

fAuthSAMLAllowDuplicateUserIDs

This variable, when set to true, enables the use of duplicate User ID's when using SAML authentication. This requires that the SAML assertion send a unique GUID or identifier for the users. The default value for this variable is false.

Please note that the use of duplicate userIDs isn't recommended.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // If set to true, duplicate UserIDs will be allowed
    // when using SAML authentication
    bp.Vars.fAuthSAMLAllowDuplicateUserIDs = true;
}
```

MatchSAMLGroups

This variable, when set to true, will, when group membership is specified in the SAML assertion, match the group assignments of an imported user to existing groups on the Process Director installation. When activated, the following import actions will occur to accomplish the group matching:

1. Users will be removed from group membership in existing groups that don't exist in the SAML assertion, **with the exception of** groups that do **not** have `AuthType` set to `SAML`.

2. Users will be added to groups of which they aren't currently a member if the group exists in the SAML assertion. Users will be added to the groups, even if the group does **not** have `AuthType` set to `SAML`.

In other words, if a Process Director User Group exists that has the same name as a SAML group contained in the assertion, but the existing group does **not** have `AuthType` set to `SAML`, the user import will ensure the user is always added to, but never removed from, the existing group.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Match SAML groups with PD group membership
    bp.Vars.MatchSAMLGroups = true;
}
```

SAML_Artifact_URL

A string variable that sets the optional Identity Provider artifact URL.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SAML Optional IDP URL
    bp.Vars.SAML_Artifact_URL = "http://www.SAMLPro-
viderURL.com";
}
```

SAML_Attrib_CustomDate

A string variable that sets the name of the attribute containing a Custom Date.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom date attribute
    bp.Vars.SAML_Attrib_CustomDate = "AttributeName";
}
```

SAML_Attrib_CustomString

A string variable that sets the name of the attribute containing a Custom String.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom string attribute
    bp.Vars.SAML_Attrib_CustomString = "AttributeName";
}
```

SAML_Attrib_CustomString2

A string variable that sets the name of the attribute containing a second Custom String.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom string 2 attribute
    bp.Vars.SAML_Attrib_CustomString2 = "AttributeName";
}
```

SAML_AuthType

This variable enables you to specify that the system will treat SAML users as if they were the Windows user type.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Treat SAML users as Windows users
    bp.Vars.SAML_AuthType = User.eAuth.Windows;
}
```

SAML_Issuer

An optional string variable that sets the ID of the SAML issuer. This ID is sometimes the same as the EntityId.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SAML Issuer ID attribute
    bp.Vars.SAML_Issuer = "IssuerID";
}
```

SAML_Attrib_CustomNumber

A string variable that sets the name of the attribute containing a Custom Number.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom number attribute
    bp.Vars.SAML_Attrib_CustomNumber = "AttributeName";
}
```

SAML_Attrib_Email

A string variable that sets the name of the attribute containing the User's email address.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Email attribute
    bp.Vars.SAML_Attrib_Email = "AttributeName";
}
```

SAML_Attrib_Groups

A string variable that sets the name of the attribute containing the User Groups.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // User groups attribute
    bp.Vars.SAML_Attrib_Groups = "AttributeName";
}
```

SAML_Attrib_GUID

A string variable that sets the name of the attribute containing the User GUID.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // GUID attribute
    bp.Vars.SAML_Attrib_GUID = "AttributeName";
}
```

SAML_Attrib_UserID

A string variable that sets the name of the attribute containing the UserID.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // User ID attribute
    bp.Vars.SAML_Attrib_UserID = "AttributeName";
}
```

SAML_Attrib_UserName

A string variable that sets the name of the attribute containing the UserName.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Username attribute
    bp.Vars.SAML_Attrib_UserName= "AttributeName";
}
```

SAML_Enable

A Boolean variable that determines whether to require that SAML login. Setting the value to "true" will require the SAML login. The default value is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Require SAML login
    bp.Vars.SAML_Enable = true;
}
```

SAML_Enable_ SLO

For Process Director v6.0.100 and higher, this Boolean variable determines whether to enable Azure Single Sign-Out for SAML. Setting the value to "true" will when logging out of Process Director, also completely log the user off the SAML Identity Provider (Azure). The default value is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable SAML Single Sign-Out for Azure
    bp.Vars.SAML_Enable_SLO = true;
}
```

SAML_IP_AssertionCertificate

A string variable that sets the optional path to the identity provider's public certificate used to validate the assertions in the SAML response.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Assertion certificate path
    bp.Vars.SAML_IP_AssertionCertificate =
     "https://www.certificateURL.com/path/certificate.cer";
}
```

SAML_IP_Certificate

A string variable that sets the optional path to the identity provider's public certificate used to validate the entire SAML response.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // IDP certificate path
    bp.Vars.SAML_IP_Certificate =
     "https://www.certificateURL.com/path/certificate.cer";
}
```

SAML_My_Certificate

A string variable that sets the optional path to **your** public certificate used for SAML requests.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Certificate path
    bp.Vars.SAML_My_Certificate =
     "https://www.certificateURL.com/path/certificate.cer";
}
```

SAML_My_PFX

A string variable that sets the optional path to the PFX file used to sign SAML requests.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // PFX file location
    bp.Vars.SAML_My_PFX = "C:\File\Path";
}
```

SAML_My_PFXPassword

A string variable that sets the optional password of the PFX file used to sign SAML requests.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // PFX file password
    bp.Vars.SAML_My_PFXPassword = "password";
}
```

SAML_NextURLInRelayState

A Boolean variable that determines whether the "next URL" is set inside the relay state variable. The default value is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable URL relay
    bp.Vars.SAML_NextURLInRelayState = true;
}
```

SAML_NoLoginButton

A Boolean variable that determines whether the SAML login button will be removed from the home page. The default value is "false". Setting the variable to "true" will hide the login button.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Hide the SAML login button
    bp.Vars.SAML_NoLoginButton = true;
}
```

SAML_ProviderName

An optional string variable that sets the ProviderName of the SAML issuer.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SAML provider name attribute
    bp.Vars.SAML_ProviderName = "ProviderName";
}
```

SAML_URL

A string variable that sets the URL of the SAML identity provider.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // SAML IDP URL
    bp.Vars.SAML_URL = "http://www.SAMLProviderURL.com";
}
```

SAML_URL_Destination

A string variable that sets the optional Destination URL of the SAML Identity Provider, i.e., the URL in SAML request Destination field.

Example

```
public override void SetSystemVars(BPLogix.WorkflowDirector.SDK.bp bp)
{
    // SAML Destination URL
    bp.Vars.SAML_URL_Destination = "http://www.SAMLProviderURL.com";
}
```

SAML_URL_Logout

A string variable that sets the optional URL of the SAML logout page to use when redirecting a user who logs out of Process Director.

Example

```
public override void SetSystemVars(BPLogix.WorkflowDirector.SDK.bp bp)
{
    // SAML logout URL
    bp.Vars.SAML_URL_Logout = "http://www.SAMLLogoutURL.com";
}
```

Social Media Custom Variables

Process Director enables you to connect to a variety of Social Media Datasources. For the purposes of this documentation, we will include other services, such as Microsoft Dynamics, even though they aren't classified as social media connections, since the connections are implemented in the same fashion as social media connections.

Social Media Connections

There are four sysvar types that you can use when creating social media connections:

1. ConnectionString: String value. The Connection string used to connect to the social media source. The connection string for each social media connector already have default values in Process Director that automatically use the

appropriate OAuth app tokens and token secrets to connect to the data source. They should not, therefore, require customization. You may, of course, customize these connection strings if necessary. For Microsoft OneDrive, Dropbox, and Box.Net, the Connection String properties aren't needed, but these data sources still implement the remaining three sysvar types below.

2. Accomplishment: Boolean value. Show the Client token to allow users to customize it.
3. DefaultAppToken: String value. The default OAuth application token for the social media source.
4. Default App Token Secret: String value. The default AOAuth token secret for the social media source.

Each of the fours sysvar types above are implemented separately for each social media source as follows:

SOCIAL MEDIA SOURCE	SYSVAR NAME
LinkedIn	LinkedInConnectionString
	LinkedInShowClientToken
	LinkedInDefaultAppToken
	LinkedInDefaultAppTokenSecret
Google Sheets	GoogleSheetsConnectionString
	GoogleShowClientToken
	GoogleDefaultAppToken
	GoogleDefaultAppTokenSecret
Dropbox	DropboxShowClientToken
	DropboxDefaultAppToken
	DropboxDefaultAppTokenSecret
Microsoft OneDrive	OneDriveShowClientToken
	OneDriveDefaultAppToken
	OneDriveDefaultAppTokenSecret

SOCIAL MEDIA SOURCE	SYSVAR NAME
Box.Net	BoxNETShowClientToken
	BoxNETDefaultAppToken
	BoxNETDefaultAppTokenSecret

In addition, for users of Federated Identity, Facebook, Twitter and Google, there are tokens and token secrets specifically for use with federated identity, and which are implemented through the following string variables:

SOCIAL MEDIA SOURCE	SYSVAR NAME
Google	GoogleDefaultSSOAppToken
	GoogleDefaultSSOAppTokenSecret

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default connection string
    bp.Vars.ShowSocialGoogleSheets = true;
    bp.Vars. GoogleSheetsConnectionString =
    "Offline=false;user={0};password={1};";
}
```

Social Media Datasource Creation

There is a set of Boolean variables that determine whether specific social media Datasources will display as available Datasource types when creating new Data-source objects in the Process Director interface. Setting the value to "false" means that the item won't be displayed in the Datasource Type dropdown in the Create Datasource screen of the Process Director interface.

The available variables are listed below.

- ShowSocialGoogleSheets
- ShowSocialAmazonDB
- ShowSocialSalesforce
- ShowSocialDropbox
- ShowSocialLinkedIn
- ShowSocialOneDrive
- ShowSocialBoxNET
- ShowDynamics

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default connection string
    bp.Vars.ShowSocialOneDrive= false;
    bp.Vars.ShowDynamics= false;
}
```

Additional Social Media variables are documented below.

AmazonDBConnectionString

This string variable enables you to customize the connection string to Amazon DB. The default value is shown in the example below.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default connection string
    bp.Vars.AmazonDBConnectionString =
      "Offline=False;Access Key={0};Secret Key={1}";
}
```

DynamicsConnectionString

This string variable enables you to customize the connection string to Microsoft Dynamics CRM. The default value is shown in the example below.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default connection string
    bp.Vars.DynamicsConnectionString = "User={0};Password={1};
     Url={2};CRMVersion=CRM 2013;";
}
```

SalesforceConnectionString

This string variable enables you to customize the connection string to SalesForce. The default value is shown in the example below.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Default connection string
    bp.Vars.SalesforceConnectionString = "User={0};Password=
{1};
     Security Token={2}";
}
```

System Custom Variables

System-Level Custom Variables can be set to control the general operation of the Process Director installation.

EmbedDocumentTypes

This variable enables you to control document types are displayed using the EMBED tag in the browser. You can inspect or modify this list.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Add the MP4 file type as one to display using the EMBED
tag
    bp.Vars.EmbedDocumentTypes.Add("mp4");
}
```

fAllowUnencodedSysvarsinBV

This variable disables the ability for users to configure a Custom Task or Business Value using SQL Commands that do **not** use the proper SQL encoding for variables contained in the SQL Statements. When set to "false", Process Director won't allow the Custom Task or Business Value to be saved until all of the variables used in the SQL statement have been given the appropriate SQL-safe encodings specified in the system Variables Reference Guide. For example, using a parameter in a Business Value's SQL statement might require the SQL-safe encoding symbol "$" in the parameter variable like this:

```
SELECT  *  FROM  Training_ Vendor  WHERE  VendorName  LIKE  '%
{$PARAMETER:Vendor}%'
```

The default value for this custom variable is "true". When set to "true", Process Director will issue a warning to the user that variables aren't properly encoded, but *will* allow them to save the configuration.

> ⚠️ **BP Logix recommends that you set this custom variable to "false", if possible. You may—indeed, probably—have existing Custom Tasks or Business Values whose variable values aren't properly encoded, so you should *not* set this value to "false" until you've ensured that all of your existing Custom Tasks and Business Values use the proper encodings.**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent saving CT or BV configurations that don't use SQL-safe
    // encodings for variable values.
    bp.Vars.fAllowUnencodedSysvarsinBV = false;
}
```

fCloseEditAfterUpload

If this system variable is set to true, the popup that appears after a check-in will close automatically when the upload completes.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // the popup appearing after a successful check-in will not
    // close while fCloseEditAfterUpload is set to false
    bp.Vars.fCloseEditAfterUpload = false;
}
```

fCopyRefsFromRealProcessToKViewProcess

When starting a process from a Knowledge View against returned form instances, the system will copy all attachment references from the original process associated with the form instance to the process instance that is being started by the Knowledge View. This enables Custom Tasks like Convert to PDF and Export Items to

work with the original documents/form instances that are in the original process. The default behavior can be disabled by setting this custom variable to "false".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable copying of original attachments when a process
    // is initiated via Knowledge View
    bp.Vars.fCopyRefsFromRealProcessToKViewProcess = false;
}
```

fDeleteDocOnRemove

When true, this option immediately deletes documents when not referenced by any other Form instance. When false, documents are marked for deletion, and actually deleted during normal clean-up processing.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Deletes documents immediately when they are removed
    bp.Vars.fDeleteDocOnRemove = true;
}
```

fDisableCSVNumberStringLogic

Prior to v5.31, when importing a CSV with numbers and strings in a column, the system, by default, looked at the first row of the CSV file to determine the data type. This could result in a data type error when the first row contains a number and subsequent rows contain character strings. For v5.31 and higher, data imported will default to the String datatype, unless a different datatype is specified in the header row. This variable, when set to "true", will disable the new logic, and revert to the pre-v5.31 conversion logic. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Revert to the old CSV field type logic
    bp.Vars.fDisableCSVNumberStringLogic = true;
}
```

fDocRenameLogicOff

This variable permits the name of a document object in the content list to be changed to match the name of the document being checked in, when set to "true". The default value is "false", i.e., that the name in the content list is retained even if the document being checked in has a different name.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Turn on object renaming
    bp.Vars.fDocRenameLogicOff = true;
}
```

fEnableContentListLimit

This variable, when set to "false", will display an unlimited number of objects that reside in a Content List folder when it is viewed. This may have serious impact on performance when navigating through the Content List. The default value for this variable is "true", and it applies the default limit (1000) for the number of items that can be returned, or, alternatively, the number of items specified in the nKViewBuiltinMaxResults custom variable, when configured.

> ⓘ **For Process Director v6.1.003 and below, the default value for this variable was "false".**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Will enable display of an unlimited number of Content
```

```
List folder items.
    // May impact performance when navigating through the Con-
tent List.
    bp.Vars.fEnableContentListLimit = false;
}
```

fEnableJavaScriptDev

This variable permits form variables to be set or retrieved from JavaScript, when set to the default value of "true". Setting this value to "false" disables this behavior.

Vars.cs Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable JavaScript access to vars
    bp.Vars.fEnableJavaScriptDev = false;
}
```

JavaScript Example

By default, JavaScript can be used to get or set form value for controls that aren't contained in a Form array. The best practice for this type of use would most often be to insert the JavaScript into an HTML Control on the Form.

```
<script>
    // Function call to get form data from a Form via
JavaScript
    function getVariable()
    {
        var frmValue = CurrentForm.FormControls["FORM_
VAR"].value;
    }


    // Function call to set a Form field value with JavaScript
    function setFieldValue()
    {
        var frmValue = "SomeValue";
        CurrentForm.FormControls["FORM_VAR"].value = frmValue;
    }
</script>
```

fEnableSQLEscape

When you run a SELECT statement that contains a LIKE operator and an ESCAPE clause in SQL Server 2008 R2, SQL Server 2012, or SQL Server 2014, SQL Server may use an inefficient query plan for the statement. Additionally, the performance of the statement is low. This SQL Server bug can cause a performance problem with Knowledge Views searching for form data using "contains", especially when using the "contains" operator with wildcard characters. This Custom variable, when set to "true", alters the operation of these queries to improve performance and help mitigate this SQL Server bug. The default value for this variable is "false".

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Mitigate SQL Server bug for LIKE/Wildcard queries
    bp.Vars.fEnableSQLEscape = true;
}
```

FileUploadBlacklist

This property accepts a comma-separated string of file extension that, when set, will prevent files with those file extensions from being uploaded via any attachment control. This property is a universal blacklist of file uploads for files with the listed extensions.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disables upload of listed file types
    bp.Vars.FileUploadBlacklist = "dotx, xltx, dot, xlt";
}
```

FileUploadBlacklistAlternateText

This property enables you to universally set the `Allowed File Alternate Text` property of Attach Object controls, instead of setting the property individually in the control.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disables upload of listed file types
    bp.Vars.FileUploadBlacklistAlternateText =
      "Word Documents, Excel Spreadsheets, Word Templates,
Excel Templates";
}
```

Locales

Process Director enables the addition of locales by editing the vars.cs file.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // If you comment out the first line,
    // it will append new values to the default list
    // Otherwise a new list will be created
    Locales = new List<NameValue>();
    Locales.Add(new NameValue("English", "en"));
}
```

ObjectLockingEnable

This boolean variable enables you to turn off object locking by setting the value to "false". The default value is "true".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will enable object locking.
    bp.Vars.ObjectLockingEnable = true;
}
```

ObjectLockingForce

This boolean variable enables you to require object locking when set to "true". The default value is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will require object locking when set to true.
    bp.Vars.ObjectLockingForce = false;
}
```

Project Reminder Times

This variable enables you to customize the default reminder times that are displayed in a Timeline Activity's Notifications tab.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Activity Reminder times
    // If you comment out this next line, it will append new
values to
    // the default list
    // Otherwise, a new list will be created
    bp.Vars.ProjectReminderTimes= new List<TimeValue>();
    bp.Vars.ProjectReminderTimes.Add(new TimeValue(-1 * (60 *
60 * 24),
      "1 Day Before Due"));
    bp.Vars.ProjectReminderTimes.Add(new TimeValue(-2 * (60 *
60 * 24),
      "2 Days Before Due"));
    bp.Vars.ProjectReminderTimes.Add(new TimeValue(1 * (60 *
60 * 4),
      "Every 4 Hours"));
    bp.Vars.ProjectReminderTimes.Add(new TimeValue(1 * (60 *
60 * 24),
      "Every Day"));
    bp.Vars.ProjectReminderTimes.Add(new TimeValue(4 * (60 *
60 * 24),
      "Every 4 Days"));
    bp.Vars.ProjectReminderTimes.Add(new TimeValue(7 * (60 *
60 * 24),
      "Every Week"));
}
```

UploadTempPath

This variable enables you to specify a folder to use as a temporary folder when using the Multi-File Upload functionality with the AttachObjects control. The default system temp path may not release large file uploads properly. Setting your own folder in the folder system can alleviate this issue.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set the temporary path for large file uploads
    CV.UploadTempPath = @"m:\tempupload";
}
```

Workflow Reminder Times

Process Director enables the reminder times in the process definition to be customized in the dropdown lists.

> ⓘ **The Workflow object has largely been deprecated by the Process Timeline. BP Logix recommends the use of the Process Timeline as the process model for all new development.**

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Step Reminder times
    // If you comment out this next line, it will append new
values to
    // the default list
    // Otherwise, a new list will be created
    bp.Vars.StepReminderTimes = new List<TimeValue>();
    bp.Vars.StepReminderTimes.Add(new TimeValue(-1 * (60 * 60
* 24),
     "1 Day Before Due"));
    bp.Vars.StepReminderTimes.Add(new TimeValue(-2 * (60 * 60
* 24),
     "2 Days Before Due"));
    bp.Vars.StepReminderTimes.Add(new TimeValue(1 * (60 * 60 *
4),
     "Every 4 Hours"));
    bp.Vars.StepReminderTimes.Add(new TimeValue(1 * (60 * 60 *
24),
     "Every Day"));
    bp.Vars.StepReminderTimes.Add(new TimeValue(4 * (60 * 60 *
24),
     "Every 4 Days"));
    bp.Vars.StepReminderTimes.Add(new TimeValue(7 * (60 * 60 *
24),
     "Every Week"));
}
```

Task Custom Variables

Custom variables in this section of the documentation can be used to customize a variety of settings associated with user tasks.

AlwaysFindTaskForForms

When this flag is set to true, forms will attempt to associate with a task when opened. If this is left as the default (false), forms will only associate with a task when opened from a task list Knowledge View.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.AlwaysFindTaskForForms = true;
}
```

fCancelSubWorkflows

If this system variable is set to false, cancelling a process or process task won't cancel their associated sub-processes.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Cancellations of tasks or processes won't cancel sub-
    // processes
    bp.Vars.fCancelSubWorkflows = false;
}
```

fEnableUndelegationRestart

When a user who is participating in a running task delegates to another user who is already running in the same step/activity, the original user is canceled and the delegated user is left running. When undelegating, this system variable, when set to "true", will allow the original user's to be restarted if that step/activity is still running for the delegated user. More information about how this variable affects user delegation is available in the User Delegation topic of the System Administrator's Guide.

This variable should be set in the PreSetSystemVars() function of the customization file.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Original user will be restarted when the delegation is
removed
    // from a running task.
    bp.Vars.fEnableUndelegationRestart = true;
}
```

fForceInviteEmail

This system variable, when set to "false", enables you to prevent the standard task assignment email notification from being sent to users who are invited to a task using the email invite feature. The default value for this variable is "true".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent assignment emails from being sent to invited
task assignees
    bp.Vars.fForceInviteEmail = false;
}
```

fPreventTaskCompleteIfCheckout

When set to true, this option prevents users from completing a task if they've checked out a document while working on that task, and haven't yet checked it in.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevents user from completing a task
    // if he performed a check out on a
    // document and hasn't checked it in
    bp.Vars.fPreventTaskCompleteIfCheckout = true;
}
```

fSendEmailOnWfAdmin

When set to true, This variable will force an email to be sent when a user is added to a running step/activity by an administrator. The default value is false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Send an email on administrative user additions to a
task
    bp.Vars.fSendEmailOnWfAdmin = true;
}
```

fShowResultOnNotNeeded

This option, when set to "true" will display the task result in the routing slip when the task is completed by an automated or external process. When a task's "Completed When" condition is "When Any Result Condition is Met" and a non-user result condition is met, (such as a check box being checked), the result of that automated or external process is shown in the routing slip. By default, this variable is set to "false".

When an activity is configured to complete when "All Users Complete or Result Condition Met", this variable will show the activity status associated with each of the users that were not needed.

This variable should be set in the PreSetSystemVars() function of the customization file.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Users marked as "not Needed" will display in the Rout-
ing Slip
    bp.Vars.fShowResultOnNotNeeded = true;
}
```

fStartUsersAddedToGroup

This option, when set to "true" will automatically add users to a running instance of a process task when the users are added to the group assigned to the task. The default setting for this custom variable is "false".

This variable should be set in the PreSetSystemVars() function of the customization file.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // New group users will automatically be added to running
tasks
    bp.Vars.fStartUsersAddedToGroup = true;
}
```

TaskAlreadyCompleteAlert

This variable enables you to configure the message displayed in the alert box that will display when the user tries to open a task that has already been completed. Set to "" if you don't want the alert to display at all.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.TaskAlreadyCompleteAlert = ""; //Alert won't dis-
play
}
```

TaskAlreadyCompleteMessage

This variable enables you to configure a text message displayed on the page that will be displayed when the user tries to open a task that has already been completed. This only applies when the page to which the user is directed resides in Process Director. Set to "" if you don't want the message to display at all.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.TaskAlreadyCompleteMessage = ""; //Message won't
display
}
```

TaskAlreadyCompletePage

This variable enables you to configure which page is displayed when the user tries to open a task that is already complete. You can set this variable to an HTML link of the page you wish displayed. By default, the home page is displayed. You can set the variable to "null" will display the process instance page. You can also specify that Process Director shows the Form associated with the task by setting the variable to "form.aspx".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Displays process instance page
    // To display the Form associated with the task use:
    // bp.Vars.TaskAlreadyCompletePage = "form.aspx";
    bp.Vars.TaskAlreadyCompletePage = null;
}
```

TaskAssignedReminderTimes

This variable enables you to add an option to the reminder dropdown in a Timeline Activity that will remind the user he has been assigned that activity a given number of seconds after it has been assigned to him.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Add a new reminder time
    bp.Vars.TaskAssignedReminderTimes.Add(new TimeValue(1 *
(10 * 60),
      "Ten minutes after task is assigned.");
}
```

Task Due Reminder Times

Task Due reminder Times are times that reminders should be sent after a task's due date has passed. In the example below, the reminder times add additional reminder options to the standard list, and once added, will also appear in the drop-down control as selectable options.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.TaskDueReminderTimes.Add(new TimeValue(-1 * (24 *
(60 * 60)),
      "Every 1 day after task is due"));
    bp.Vars.TaskDueReminderTimes.Add(new TimeValue(-2 * (24 *
(60 * 60)),
      "Every 2 days after task is due"));
    bp.Vars.TaskDueReminderTimes.Add(new TimeValue(-3 * (24 *
(60 * 60)),
      "Every 3 days after task is due"));
```

```
    bp.Vars.TaskDueReminderTimes.Add(new TimeValue(1 * (10 *
  60),
      "Ten minutes after task is due.");
    bp.Vars.TaskDueReminderTimes.Add(new TimeValue(-1 * (10 *
  60),
      "Every ten minutes after task is due.");
}
```

User Info SlideOut Custom Variables

The User Info Slideout is a control that appears when you click the upper left corner of the Process Director screen. It contains a variety of information about the logged-in user, and can include the Name, email address, image, and other items that can be customized by using the custom variables in this section of the documentation.
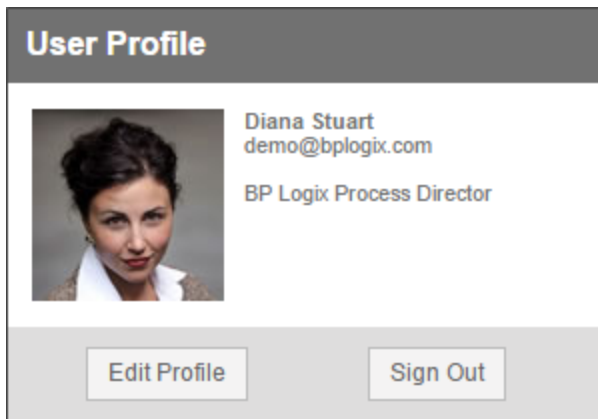
fDisableUserProfileEmailChange

This variable, when set to "true" disables all end users' ability to change their email addresses via the User Profile page. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent users from changing their email addresses
    bp.Vars.fDisableUserProfileEmailChange = true;
}
```

fTurnOffUserProfileTimeZone

By default, users are allowed to control the time zone setting in their user profile. Setting this variable to "true" disables this ability.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent users from setting time zones
    bp.Vars.fTurnOffUserProfileTimeZone= true;
}
```

UserInfoShowSignOut

Setting this system variable to true enables you to show the Sign Out button on the user info slide panel.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Hide the user info slideout
    bp.Vars.UserInfoShowSignOut = false;
}
```

fTurnOffDelegation

This variable , when set to "true" will remove the ability for users to perform delegation from their User Profile page. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will disable user delegation in the user profile.
    bp.Vars.fTurnOffDelegation = true;
}
```

fTurnOffSharedDelegation

This variable , when set to "true" will remove the ability for users to perform shared delegation from their user profile page. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will disable shared delegation ability in the user
profile.
    bp.Vars.fTurnOffSharedDelegation = true;
}
```

fTurnOffUserProfileEmail

By default, users are allowed to disable their email from their user profile. Setting this variable to "true" disables this ability.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent users from disabling email
    bp.Vars.fTurnOffUserProfileEmail= true;
}
```

UserInfoSlideOut

This system variable determines what information will be displayed in the user info slide panel. The system variable can contain HTML and system variables.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Format the user info slideout
    bp.Vars.UserInfoSlideOut = "<span style='font-
weight:bold;'>
    {Curr_User,format=name}</span><br/>{Curr_User-
,format=email}
    <br/><br/>{server_name}";
}
```

UserInfoShowEditProfile

This system variable enables you to determine whether the "Edit Profile" button will display in the user info slide panel. If this variable is set to true, the button will be displayed. Otherwise, it will not.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable user edits
    bp.Vars.UserInfoShowEditProfile = false;
}
```

User Interface Custom Variables

These Custom Variables control the appearance of the Process Director's user interface for end users.

AllowRichTextTemplate

When this flag is set to true, Process Director will attempt to transfer Rich Text content into PDF form fields that are set to accept Rich Text content. There are limitations to this ability, which are explained in the PDF Rich Text Field Support section of the PDF Custom Tasks topic.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.AllowRichTextTemplate = true;
}
```

AutoMultilineTextBoxResize

By default, an Input control placed on a Form, whose Rows property is set to the default setting of "1", will be displayed at run-time as an HTML Text control. Setting the Rows property to a value of "2" or higher, will, at run-time, convert the control to an HTML TextArea control. This variable, when set to "true", will place a resizing handle on all TextArea controls. Users will be able to resize the control to the height and width they desire while entering data. Otherwise, the control's size will remain fixed, and scrollbars will appear as more data is entered than can be vertically displayed in the control.

In addition, this variable, in conjunction with the AutoMultilineTextBoxResizeClass Custom Variable, can enable the automatic resizing of TextArea controls as you type. A default CSS class, BPExpanding, is designed to

automatically resize a TextArea while you type, by automatically expanding the control vertically for each new line, as needed. The example below shows the configuration you'll need to add to the customization file in order to enable automatic resizing.

Once you've configured both the `AutoMultilineTextBoxResize` and `AutoMultilineTextBoxResizeClass` variable, you can enable the feature in each Input control by setting the control's CSS Class property to BPExpanding. Controls without this CSS Class property setting won't automatically expand.

If you do not add the `AutoMultilineTextBoxResizeClass` variable to enable auto-resizing, the TextArea control will still be manually resizable when `AutoMultilineTextBoxResize` is set to True.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable resizing for multi-line text boxes
    bp.Vars.AutoMultilineTextBoxResize = true;

    // Invoke the default class to enable auto-resizing
    bp.Vars.AutoMultilineTextBoxResizeClass = "BPExpanding";
}
```

AutoMultilineTextBoxResizeClass

By default, Process Director uses a CSS class named BPexpanding to implement auto-resizing for Input controls. You can change the default class to a class of your own design, by setting the class name with this custom variable.

Each Input control that you wish to auto-resize must use the name of your custom class as its CSS Class property, instead of BPExpanding.

> (i) **Use of this custom variable requires that the `AutoMultilineTextBoxResize` variable be set to "true".**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable autosized text boxes
bp.Vars.AutoMultilineTextBoxResize = true;

    // Use a custom class to enable it
    bp.Vars.AutoMultilineTextBoxResizeClass = "myCus-
tomCSSClass";
}
```

bpFormOpenSize

Enables you to set custom sizes / position for forms that are opened.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // All forms should open at position 10,10 with size
700,600
    bp.Vars.bpFormOpenSize = "top-
p=10,left=10,height=700,width=600";
}
```

bpPopupOpenSize

Enables you to set custom sizes / position for all popup windows such as user pick-ers.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // All popups should open at position 10,10 with size
700,600
    bp.Vars.bpPopupOpenSize = "top-
p=10,left=10,height=700,width=600";
}
```

CustomHTMLHeadTags

This variable enables you to enter an HTML string to define custom Head tags for the HTML pages displayed in Process Director.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.CustomHTMLHeadTags = "<link href-
f=\"http://sample.bplogix.com/custom
    /sample-bp-icon.jpg\"
    rel=\"apple-touch-icon\"><link rel=\"stylesheet\"
    type=\"text/css\" href=\"theme.css\">";
}
```

DisableInlineErrorsWithPopup

This variable, when set to "true", suppresses the inline form validation error messages on a Form when using the option to display error messages in a popup. The default for this option is "False".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Suppress inline error messages when using Popup error
messages
    bp.Vars.DisableInlineErrorsWithPopup = true;
}
```

DisableParentRefreshForm

This variable enables you to disable the auto refreshing of the parent browser when forms are completed.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.DisableParentRefreshForm = true;
}
```

EnableReactAdminPages

This variable was implemented in Process Director v5.44.600. When set to "false", it disables the redesigned look of some IT Admin pages to display the classic view. The default value for this variable is "true".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable admin page UI updates.
    bp.Vars.EnableReactAdminPages = false;
}
```

ErrorPage

This string variable enables you to set the URL of a custom error page to display when a Process Director error occurs. The custom error page should be placed into the /custom folder at the website root to ensure that they aren't overwritten during an upgrade of the product. An HTML and ASPX sample error page are included in the /custom folder by default.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the URL of the custom error page.
    bp.Vars.ErrorPage = "http://server-
name.com/custom/error.htm";
}
```

fEnableAccessibility

Available in Process Director v5.34 or higher, this Boolean variable, when set to "true," enables advanced accessibility features for Process Director. The default value for this variable is "false".

Advanced accessibility features include:

1. Improved contrast
2. Improved hover and focus indicators
3. Increased font sizes
4. Addition of structural element (main)
5. Proper use of heading tags (h1, h2...)
6. Proper use of the alt attribute
7. Keyboard support for images that had only mouse support
8. Setting default language in html tag

9. Use HTML5 specification for document type
10. Make error message easier to locate by added aria-describedby attribute

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable automatic accesibility features
    bp.Vars.fEnableAccessibility = true;
}
```

fIncludeBootstrap

This variable, when set to true, enables you to use the Bootstrap controls in the Online Form Designer's Responsive Layout menu, and provides visual markers on the design surface for the Bootstrap controls. The default setting for this variable is false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable Bootstrap in the Online Form Designer
    bp.Vars.fIncludeBootstrap = true;
}
```

EnableFormThemes

This variable, when set to "false" enables the Dropdown control to properly display type-ahead functionality. The default value is "true".

> ⓘ **This variable was previously used to enable Telerik themes for forms, but this functionality has been deprecated.**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Revert to unthemed operation, and enable
    // dropdown type-ahead.
    bp.Vars.EnableFormThemes = false;
}
```

fDisableDetailedAttach

This variable enables you to control whether the show the detailed option for attaching documents to forms.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Disable the detailed attach on upload
    bp.Vars.fDisableDetailedAttach = true;
}
```

fDisableImageResize

When true, this option disables automatic image resizing and rotating. Images are, by default, resized and rotated to display properly on mobile devices and when embedded in frames.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // disables automatic image resizing and rotating
    bp.Vars.fDisableImageResize = true;
}
```

Occasionally, When using Process Director, you may encounter a generic GDI+ error when attempting to display a PNG file.Setting fDisableImageResize to false will prevent the error from occurring.

fOpenFormNewTab

Available in Process Director v6.1.400 and higher, this variable, when set to true, opens Forms in a new browser tab, instead of a new browser pop-up window.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Open Forms in a new browser tab
    bp.Vars.fOpenFormNewTab = true;
}
```

sDisableNavigationScroll

This variable enables you to disable the system navigation scrolling for navigation buttons in a Workspace's navigation bar. This variable is set to false by default.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will disable the scrolling of system navigation
    bp.Vars.sDisableNavigationScroll = true;
}
```

fDocRemovedInPopup

This variable changes the system behavior when the user clicks on the download link on a document that has had its binary data removed, but the document artifact remains in the system. The default behavior is to replace the currently displayed Form with a custom error page in the same window.

When setting this variable to "true", the custom error page identified in the sCustomURL_DocRemoved variable will open in a new window instead of replacing the current Form in the same window.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Cause the page that displayed the document error
    // message to occur in a popup, the default is false
    bp.Vars.fDocRemovedInPopup = true;
}
```

fEnableKViewFilterOnSavedForLater

By default, incomplete forms that have been saved for later display in Task List Knowledge Views, irrespective of filters that may be applied to the Knowledge View. This Boolean variable, when set to "true," enables Process Director to filter incomplete forms and hide them from the Task List results.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable Task List filtering for saved/incomplete forms
```

```
        bp.Vars.fEnableKViewFilterOnSavedForLater = true;
}
```

fEnableMultiLanguage

By default, Label text values are static once set. For users of Process Director v5.44.500 and higher, this Custom Variable will, when set to True, enable dynamic setting of Label text. This feature is primarily intended to support accessibility for multilingual user interfaces where the Label text must change based on the user's language.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable dynamic text for labels
    bp.Vars.fEnableMultiLanguage = true;
}
```

fEnableThumbnails

This Boolean variable enables thumbnails of attached files to be shown in ShowAttachment Controls. The default value of this flag is "true". When set to "false", Process Director will no longer show options on the ShowAttach Form control to display the thumbnail for supported document types.

You can optionally set the height and width of the thumbnails by setting ThumbnailWidth and/or ThumbnailHeight to the desired dimensions in pixels.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enables thumbnails for attachments
    bp.Vars.fEnableThumbnails = true;

    // Sets thumbnail width to display in pixels
    bp.Vars.ThumbnailWidth = 75;
}
```

fIgnoreAccessibilityFlag

This system variable enables Process Director to ignore some settings for the Switch control, to improve accessibility for that control. This variable overrides the default fEnableAccessibility setting for the control.

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Ignore accessibility setting to Display the Switch con-
trol
    bp.Vars.fIgnoreAccessibilityFlag = true;
}
```

fLoginBgRand

This variable , when set to "true" will display a random background image on the login page. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Enable random background images for login page
    bp.Vars.fLoginBgRand = true;
}
```

FormEditorConfig

Users of Process Director v5.0 and higher have the ability to use the Online Form Designer (OFD) to design Forms. The toolbars that appear in the OFD can be customized by creating a JavaScript configuration file that you can upload to the `/cus-tom` folder of the Process Director installation. Documentation for the CKEditor's Toolbar configuration can be found at the CKEditor Documentation web site.

This variable specifies the location of a custom JavaScript configuration file to use, in addition to the default file, and primarily enables you to customize the fonts used by and displayed in the OFD. This variable will universally effect the available controls for every form definition.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set a custom configuration file for the Form Editor
    bp.Vars.FormEditorConfig = "/custom/cust_config.js";
}
```

Notice that you need to use full client path, without using a "~", in the URL to the Configuration file.

You must create a cust_config.js file in the Custom folder of your Process Director Installation.

For more information about how to perform customization of the OFD, please see the UI Customization topic.

fShowDisabledUsers

When set to true, Process Director will display disabled users in all dialogs and user pickers. This variable is set to false by default.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Display disabled users
    bp.Vars.fShowDisabledUsers = true;
}
```

fShowPredictedDates

This boolean variable sets whether to hide the predicted start/end dates from the routing slip. The default value is false.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Hide predicted dates on the Routing Slip
    bp.Vars.fShowPredictedDates = false;
}
```

fShowProcessCancelReasonOnUser

When a process or activity/step is canceled, and that results in a user being canceled, the administrative comment will be added to the user record in the Routing Slip indicating why it was canceled if this variable is set to "true". The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Show cancellation reason on Routing Slip
    bp.Vars.fShowProcessCancelReasonOnUser = true;
}
```

InlineDocumentTypes

This variable enables you to control document types are displayed "inline" in the browser. You can inspect or modify this list.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Add the ZIP file type as one to display "inline"
    bp.Vars.InlineDocumentTypes.Add("ZIP");
}
```

LeaveCaseButtonText

This variable enables you to change the default text used in the User Info Box to close a case folder. By setting the variable, you can change the default "Leave Case" text to custom text of your choice.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Change text of the Leave Case button
    bp.Vars.LeaveCaseButtonText= "Leave Case text";
}
```

LoginMessage

This variable enables you to set a string on the login page. This string can contain HTML syntax.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set HTML message on login page
    bp.Vars.LoginMessage = "Welcome to Process Director.
     <a href='http://bplogix.com/' "+
    "target='_blank'>Click Here</a>for the BP Logix web
site.";
}
```

nFormOpenProps

This variable enables you to set an option to open a form in normal, maximized or full screen mode.

Values

VALUE NAME	DESCRIPTION	DEFAULT
FormOpenProps.Normal	Comes up as a popup window smaller than the screen size	
FormOpenProps.Maximized	Comes up as a popup sized to the full screen	Default
FormOpenProps.UseFullScreen	Uses the browser full screen para-meter to max-imize the popup window	

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Sets Forms to open in a normal window
    bp.Vars.nFormOpenProps = FormOpenProps.Normal;
}
```

nHomeTopHeight

This variable enables you to set the height of the navigation bar on the home page in pixels. This may be needed if you customize the logo.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set top home bar height to 60 pixels
    bp.Vars.nHomeTopHeight = 60;
}
```

nHomeTopWidth

This variable enables you to set the width of the custom logo on the home page in pixels.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Set custom logo width to 200 pixels
    bp.Vars.nHomeTopWidth = 200;
}
```

nPDFPageWidth

This variable enables you to control the width in pixels of forms converted into PDF. 0 sets the width automatically.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // 1024 pixel width for converted PDFs
    bp.Vars.nPDFPageWidth = 1024;
}
```

nTaskCompleteDialogWidth

This variable sets the width of the "Completing Task" popup that appears when a task is completed from the Task List. In most cases, this setting will never need to be adjusted.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // The width of the popup that says "completing task".
    bp.Vars.nTaskCompleteDialogWidth = 20;
}
```

nTaskCompleteDialogHeight

This variable sets the height of the "Completing Task" popup that appears when a task is completed from the Task List. In most cases, this setting will never need to be adjusted.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // The height of the popup that says "completing task".
    bp.Vars.nTaskCompleteDialogHeight = 20;
}
```

nTaskCompletePromptDialogWidth

This variable sets the width of the dialog box that prompts the user to enter task completion comments when a task is completed from the Task List. In most cases, this setting will never need to be adjusted.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // The width of the popup that PROMPTS the user to enter
completion comments.
    bp.Vars.nTaskCompletePropmtDialogWidth= 500;
}
```

nTaskCompletePromptDialogHeight

This variable sets the height of the dialog box that prompts the user to enter task completion comments when a task is completed from the Task List. In most cases, this setting will never need to be adjusted.

Example

```
public override void SetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // The width of the popup that PROMPTS the user to enter
completion comments.
    bp.Vars.nTaskCompletePromptDialogHeight= 250;
}
```

NTLM_NoLoginButton

This variable enables you to remove the NTLM login button on the home page.

Example

```
public override void PreSetSystemVars (BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Removes the Windows login button
    bp.Vars.NTLM_NoLoginButton = true;
}
```

ResponsiveType

This variable enables improved responsive browser support for Forms. Enabling responsive form support activates a number of features to make Forms display better on small screens, including the ArrayColumn Form control tag that reformats array tables so that each column displays on a new row in small viewports, eliminating the need for horizontal scrolling.

The following options are available:

- `ResponsiveTypes.None`: Disables responsive support.
- `ResponsiveTypes.Mobile`: Enables responsive support only on mobile devices.
- `ResponsiveTypes.MobileSmall`: Enables responsive support only on small mobile devices, e.g., smartphones, but not iPads.
- `ResponsiveTypes.All`: Enables responsive support on all devices.

The default parameter for this variable is `ResponsiveTypes.All` for Process Director v5.34 and higher. For prior versions of the product, the default value is `ResponsiveTypes.None`.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{

    // This will enable responsive formatting only on mobile
devices
    bp.Vars.ResponsiveType = ResponsiveTypes.Mobile;

    // This will enable responsive formatting on all devices.
    // It uses current window size to make decisions on format-
ting.
    bp.Vars.ResponsiveType = ResponsiveTypes.All;

    // This disables responsive support
    bp.Vars.ResponsiveType = ResponsiveTypes.None;
}
```

sCkEditorCustomConfig

Users of Process Director v5.0 and higher have the ability to use the Text Editor on Input controls, in addition to the Rich Text editor that is available in previous versions of the product. The Text Editor is a third-party control called CKEditor. The toolbars that appear in the Text Editor can be customized by creating a JavaScript configuration file that you can upload to the `/custom` folder of the Process Director installation. Documentation for the CKEditor's Toolbar configuration can be found at the CKEditor Documentation web site.

This system variable enables you to specify the location of the configuration file you wish to use to customize the Text Editor's toolbars.

Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Specifies the location of the CKEditor config file
    bp.Vars.sCkEditorCustomConfig = "/cus-
tom/js/myCKEditorConfig.js";
}
```

For more information about this customization, please see the UI Customization topic.

sCustomURL_DocRemoved

This string variable enables you to provide a custom URL to which the user should be redirected when trying to navigate to a document that has been removed from a process.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will send the user to a custom error URL when try-
ing to
    // open a document that has been removed from the system.
    bp.Vars.sCustomURL_DocRemoved = "http://server-
name/customUrl.htm";
}
```

sLoadingImage

This variable enables you customize the loading image that is displayed when the system is waiting on a Form event. The preferred file format for the image should be an animated GIF image.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Custom loading image
    bp.Vars.sLoadingImage = "http://myserver/image.gif";
}
```

sLogoLink

This variable enables you customize the destination address if the logo is clicked.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Customize logo click URL
    bp.Vars.sLogoLink = "http://myserver/my_intranet";
}
```

sLogoURL

This variable enables you customize the top left logo displayed on the home page. Use a standard URL.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Customize logo image
    bp.Vars.sLogoURL = "http://myserver/mylogo.gif";
}
```

SplitterWidth

This variable sets the width, in pixels, of the splitter bars that appear between portlets in a workspace. You can use this variable to set a custom width for the splitter bars.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Sets the width of portlet splitter bars
    bp.Vars.SplitterWidth = "10px";
}
```

ShowDocHistoryWhenDisabled

This variable, when set to "true", enables the history tab for document attachments to display, even if the ShowAttach control is disabled. The default for this option is "False".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Shows the history for attachments when the ShowAttach
control
    // is disabled.
    bp.Vars.ShowDocHistoryWhenDisabled = true;
}
```

sStyleEnabled

This variable enables you to set the system default style for enabled form fields.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the default style for Enabled form fields
to a
    // white background
    bp.Vars.sStyleEnabled = "background-color:White;";
}
```

sStyleDisabled

This variable enables you to set the system default style for disabled form fields.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the default style for disabled form
fields to a
    // gray background
    bp.Vars.sStyleDisabled = "background-color:#CCCCCC;";
}
```

sStyleError

This variable enables you to set the system default style for form fields in an error state.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the default style for form fields in an
error
    // state to a red background
    bp.Vars.sStyleError = "background-color:Red;";
}
```

sStyleRequired

This variable enables you to set the system default style for required form fields.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the default required style for form
fields to a
    // yellow background
    bp.Vars.sStyleRequired = "background-color:#FFFF99;";
}
```

sUseCSS

This variable enables you to specify a CSS file containing definitions for CSS classes. Forms will be stylized according to the code in the specified CSS files.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Adds two custom CSS Files
    bp.Vars.sUseCSS.Add("~/custom/myStyles.css");
    bp.Vars.sUseCSS.Add("~/custom/myStyles1.css");
}
```

For more information about system customization, please see the UI Customization topic.

UseWorkspaceHome

When set to true, this system variable will use the Desktop Workspace layout for the user's home page.

> ⊘ **The use of this variable was deprecated in Process Director v4.5.**

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Deprecated
    bp.Vars.UseWorkspaceHome = true;
}
```

Workflow Step Colors

Process Director enables the colors for Process Timeline Activities and Activity Result buttons to be customized in the dropdown list of colors. You can add colors to the existing list of colors, or you can create an entirely new list of your own devising.

To create your own list, start by adding the following lines of code to the PreSetSystemVars method in the Custom Vars file:

```
bp.Vars.WorkflowColors     =     new     List<ColorValue>  ();
bp.Vars.WorkflowColors.Add(new ColorValue("Default", "", ""));
```

This will create an entirely new list with a default color value, which will be the standard default color in the interface, e.g., buttons will be the standard gray color. You can then add additional colors to your list using the color addition code described below.

> ⓘ **If you merely want to add colors to the existing default color list, don't include these lines.**

To add a new colors to the existing list, or to the new list you created above, add the following line for each desired color:

```
bp.Vars.WorkflowColors.Add (new  ColorValue ("Label",  "BackColor", "ForeColor"));
```

The following Parameters are required for this line of code:

Label: The label that will appear for the color in the dropdown.

BackColor: The HTML hexadecimal, HTML named color, or RGB background color of the dropdown item.

ForeColor: The HTML hexadecimal, HTML named color, or RGB foreground color of the dropdown item.

So, using the following line of code:

```
bp.Vars.WorkflowColors.Add   (new      ColorValue   ("Cornflower", "#739CCB", "#FFFFFF"));
```

...will result in the following addition to the bottom of the color dropdown:
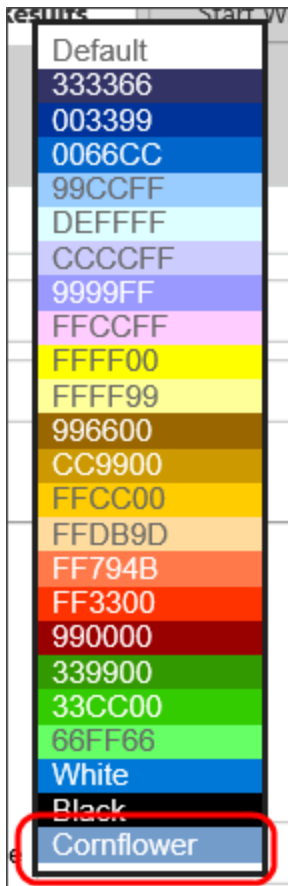
Example

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Activity colors
    // If you add the line, it will create a new color list
    // to replace the default list
    bp.Vars.WorkflowColors = new List<ColorValue>();

    // Create a new default color entry for the new list.
    bp.Vars.WorkflowColors.Add(new ColorValue("Default", "",
""));
    // If you do NOT add these lines, the colors below will
    // simply be added to the default color list.

    //Add colors to the list
    bp.Vars.WorkflowColors.Add(new ColorValue("Lt Black",
"#333366", "#FFFFFF"));
    bp.Vars.WorkflowColors.Add(new ColorValue("Lt Lavender",
"#FFCCFF", "#666666"));
    bp.Vars.WorkflowColors.Add(new ColorValue("Yellow",
```

```
"#FFFF00", "#666666"));
      bp.Vars.WorkflowColors.Add(new ColorValue("Dk Gray",
"#FFFF99", "#666666"));
}
```

> ⓘ **This variable will also accept RGB color values, e.g., "rgb(255,0,0,)", in addition to HTML color values.**

User Custom Variables

These Custom Variables enable you to specify various user-related settings in Process Director.

DefaultNewUsersToDayPass

This Custom Variable, when set to "true", will automatically create all new users as Day Pass users. The default value of this variable is "false". This variable is relevant only if you have the Day Passes license component.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Make all new users day pass users.
    bp.Vars.DefaultNewUsersToDayPass = true;
}
```

DelegationAdminGroups

If a user is in any of the groups specified by this variable, the user will be able to access the delegation administration page, even without the "User Admin" setting enabled in his user profile.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    bp.Vars.DelegationAdminGroups = "Delegation Users";
}
```

fAllowLoginRememberMe

This variable enables you to control whether built-in users can ask Process Director to remember their session so they don't have to log in each time they visit Process Director.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Allow users to use 'Remember Me' option
    bp.Vars.fAllowLoginRememberMe = true;
}
```

fAllowRetrievePassword

This variable enables you to control whether built-in users can retrieve their passwords if they are forgotten. If set to true, a prompt will appear on the login page. Development systems that use the TestUserEmailAddress won't send password reset request email to the TestUserEmailAddress, but will, instead, send them to the requesting user.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Allow users to retrieve passwords
    bp.Vars.fAllowRetrievePassword = true;
}
```

fDisableImplicitPartitionGroupUsers

Users in groups are automatically added to a partition when the group is added to the partition. They are implicitly added through their group membership. This option, when set to "true" will prevent this implicit user addition, which means that users will have to be explicitly added to a partition to be part of it.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    //Disables adding users to a partition implicitly.
    bp.Vars.fDisableImplicitPartitionGroupUsers= true;
}
```

fDisableUserRenameOnDisable

This variable will, when set to "true", prevent a disabled user from being renamed and preventing a new GUID from being applied during an update/synchronization. The default value for this variable is "false".

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Prevent disabled users from being renamed
    bp.Vars.fDisableUserRenameOnDisable = true;
}
```

fSharedDelegationNextTask

Setting this variable to true will, when shared delegation is enabled, display the next task to the delegate automatically, if the principal assignee of the current task is also the principal assignee of the next task. This variable is set to false by default.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Automatically display the next task to the delegate
    bp.Vars.fSharedDelegationNextTask = true;
}
```

fTurnOnDelegationGroups

By default, users are allowed to delegate tasks to any other user. Setting this variable to "true" restricts the user's ability to set delegations, and limits delegations only to other users who belong to the same group as the delegating user.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Restrict delegation to the same group as delegator
    bp.Vars.fTurnOnDelegationGroups = true;
}
```

nUserInactivityTimeoutSecs

This variable enables you to set the maximum number of seconds to elapse prior to automatically logging out a user for inactivity. Once the configured time limit expires, the user will be logged off, and will be forced to re-authenticate to access the system again.

Example

```
public override void SetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // This will set the maximum inactivity time to 2 hours.
    bp.Vars.nUserInactivityTimeoutSecs = 60*60*2;
}
```

UI Customization

⚠ **Please be aware that this topic is provided for informational purposes, and the methods described here should be implemented only by those with the appropriate experience with HTML, CSS, and JavaScript.**

Process Director's user interface can be customized in many ways. In this topic we'll discuss the basic process for customizing elements of the Process Director interface.

One of the most common customization methods is to use a custom stylesheet to change the display of various UI elements.

Process Director has an very extensive style sheet, named **bpw.css**, and, depending what version of Process Director you're running, a different version of this file will be used to style your installation. You should never edit this stylesheet, because you might seriously compromise the look and feel of the Process Director installation. Moreover, this stylesheet will be overwritten every time you upgrade or reinstall Process Director, so none of the changes you make would be persistent.

Instead, you can create a custom stylesheet that overrides the styles that are defined in bpw.css. This stylesheet can be placed somewhere in the `/custom` folder to make any changes persistent, since, again, this folder isn't overwritten during upgrade or reinstall. Moreover, the custom stylesheet will be much smaller, since you're only trying to modify a few specific built-in Process Director styles, in most cases, which makes creating it much simpler.

Creating this custom stylesheet will require some familiarity the bpw.css file, so you might find it useful to download it from your installation. This can be done relatively simply by using the Developer Tools built into your web browser. You can press the [F12] key to open the Developer Tools, which will give you access to all of the source files for a Process Director page, from which you can download bpw.css.

For all of the examples presented in this topic, we'll use a custom CSS stylesheet named **custom.css**, and we'll place it in the `/custom/ui` folder, so that all of our UI customization files will be organizaed in a single subfolder of the `/custom` folder.
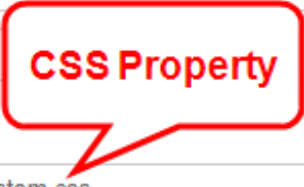
Adding a Custom CSS Stylesheet to Process Director

There are two methods for adding a custom CSS stylesheet to Process Director.

Installation Settings

On the Properties page of the Installation Settings section if the IT Admin area, the CSS property will accept the file path of a custom CSS file.

Alternatively, you can edit the Custom Variables file to add the sUseCSS Custom Variable

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Use a Custom CSS stylesheet
    bp.Vars.sUseCSS.Add("~/custom/ui/custom.css"); // We need
the "~" here
}
```

ⓘ **Note the use of the tilde (~) character at the beginning of the file location for both the Properties page and the sUseCSS Custom Variable. This**

s a required shorthand character to prompt the server add in the first
part of the fully qualified URL of the file location in Process Director.

Customizing CKEditor

For Process Director v5.00 and higher, a third-party tool, CKEditor, is used to
provide the form design and text formatting tools used in the Online Form
Designer (OFD) and in Input controls that are configured with the Use Text Editor
property selected when placed on a Form.

By default, the appearance of the CKEditor component is controlled by con-
figuration files that are installed with Process Director. This default configuration
can be overridden by the use of custom configuration files that you can add to
your Process Director Installation. Both the OFD and the Input control use different
configuration files, and each of these files can be overridden via the use of System
Variables that can be placed in the `PreSetSystemVars` portion of the custom vars
file, which is located at `/custom/vars.cs.ascx`. The `/custom` folder of a Process
Director installation not overwritten on upgrades, so any customization changes
made to the system always need to be stored in this folder, to ensure the cus-
tomization is persistent across versions.

The FormEditorConfig Custom Variable enables you to customize the OFD, while
the sCkEditorCustomConfig Custom Variable enables you to customize the Input
control's Text Editor.

> **⊘ If you make a mistake in the JavaScript configuration file for CKEd-
> itor, you can always remove the FormEditorConfig or sCkEd-
> itorCustomConfig setting from your Custom Variables file and return to
> the default Process Director settings. None of the settings you configure
> in a custom CKEditor configuration file are permanent, and can always be
> rolled back to the Process Director default.**

In both cases, a JavaScript configuration file needs to be specified for each Custom
Variable. Some knowledge of basic JavaScript syntax is required to use the con-
figuration file correctly. Happily, the syntax used in the configuration files is very
standardized, so it doesn't require too much effort to understand it.

We'll also be providing extensive examples here, so you can largely copy and paste the examples right from this documentation topic. Also, full documentation for the CKEditor's Toolbar configuration can be found at the CKEditor Documentation web site.

If you wished to customize both the OFD and the display of the Input control's Text Editor, you'd need to add the following lines of code to `PresetSystemVars`:

```
public override void PreSetSystemVars(BPLo-
gix.WorkflowDirector.SDK.bp bp)
{
    // Change the config of the Input control's text editor
    bp.Vars.sCkEditorCustomConfig = "/custom/ui/cust_txt_con-
fig.js";
    // Change the config of the Online Form Designer
    bp.Vars.FormEditorConfig = "/custom/ui/cust_ofd_
config.js";
}
```

> ⓘ **You need to use the full client path, without using a "~", in the URL to the Configuration file.**

Note that, in the examples above, both of the customization files reside in the `/cus-tom/ui` folder. Creating them in a subfolder helps to keep the system organized by placing all of your UI customization files in the same folder. You could, of course, place them in the `/custom` folder directly, but BP Logix recommends that you organize subfolders for your customizations, to keep the `/custom` folder from getting too cluttered. Note also that, in this example, the names of the JavaScript files specifically identify whether the configuration refers to the Text Editor or the OFD.

Process Director has an existing configuration file for the CKEditor built into the product. You won't ever edit that configuration file. Using a custom configuration file will simply override your installation's default configuration with your custom configuration from the customization file, but will never alter any of the built-in default settings that BP Logix has configured for the CKEditor in the Online Form Designer.

All of the customization files for CKEditor must have the `CKEDITOR.editorConfig` function. It's the only function you'll place in the file, and all customization commands must be placed insideit, as shown in the example below.

```
CKEDITOR.editorConfig = function (config) {
    // All customization commands go here.
}
```

Customizing the Online Form Designer

> ⛔ **To see the changes you make to the configuration, you'll need to delete your browser cache and reload the Process Director web page every time you upload changed versions of vars.cs.ascx or any of the customization files discussed in this topic.**

In most cases where you might want to edit the tools that are displayed in the OFD, you'll only want to remove certain tools that are of minimal use, or which you don't want Form designers to access. The `config.removeButtons` command enables you to list the buttons you want to remove from the OFD editor.

For instance, if you wanted to remove the iFrame, Smiley Face, and View Source buttons from the OFD toolbars, your custom configuration file would contain only the JavaScript below:

```
CKEDITOR.editorConfig = function (config) {
    config.removeButtons = 'iFrame,Smiley,Source';
}
```

This configuration file would simply override the default configuration file to remove the specified buttons. All the other default configuration settings would be applied, since your custom configuration file only overrides the settings of those three buttons. This is a much simpler solution for removing buttons than trying to replicate the entire configuration.

Full Customization

The default configuration for the OFD looks something like the example below. The `config.toolbar` command enables you to build the toolbars and tools you'd like to use in the OFD.

```
CKEDITOR.editorConfig = function (config) {
// Define changes to default configuration here.
// For complete reference see:
// http://docs.ckeditor.com/#!/api/CKEDITOR.config
// Toolbar default configuration as set in /Editor/bp_
```

```
config.js
// These are all the tools that are available in the default
setup of the OFD
    config.toolbar = [

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!
        // DO NOT EDIT THIS SECTION!
        // These are BP Logix Custom Tools.
        { name: 'document', items: ['BPSave', 'BPSaveOnly',
'BPSaveRun',
            'BPSaveTest', 'Bp-discard'] },

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!
        // Below are the built tools available via CKEditor
that can be edited safely
        { name: 'clipboard', items: ['Cut', 'Copy', 'Paste',
'PasteText',
            'PasteFromWord', '-', 'Undo', 'Redo'] },
        { name: 'editing', items: ['Find', 'Replace', '-',
'SelectAll', '-'] },
        { name: 'tools', items: ['ShowBlocks'] },
        { name: 'links', items: ['Link', 'Unlink', 'Anchor']
},
        { name: 'insert', items: ['Image', 'Table', 'Spe-
cialChar', 'Iframe', 'Smiley'] },
        { name: 'document', items: ['Source'] },
        '/',
        { name: 'basicstyles', items: ['Bold', 'Italic',
'Underline', 'Strike',
        'Subscript', 'Superscript', '-', 'CopyFormatting',
'RemoveFormat'] },
        { name: 'styles', items: ['Format', 'Font',
'FontSize'] },
        { name: 'paragraph', items: ['NumberedList', 'Bul-
letedList', '-', 'Outdent',
            'Indent', '-', 'Blockquote', 'CreateDiv', '-',
'JustifyLeft', 'JustifyCenter',
            'JustifyRight', 'JustifyBlock', '-', 'BidiLtr',
'BidiRtl',] },
        { name: 'colors', items: ['TextColor', 'BGColor'] },

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!
        // DO NOT EDIT THIS SECTION!
        '/',
        '/',
        // These are the common BP Logix control Tools.
```

```
        { name: 'common', items: [
            'bp-input-btn', 'bp-checkbox-btn', 'bp-datepicker-
btn', 'bp-dropdown-btn',
            'bp-radio-btn', 'bp-section-btn', 'bp-switch-btn'
            ]
        },
        {
            // These are the remaining BP Logix control tools
            // The items below are hidden in bpw.css, which
can't be edited.
            name: 'bplogix', items: [
                'Bp-attach', 'Bp-input', 'Bp-checkbox', 'Bp-
datepicker',
                'Bp-dropdown', 'Bp-radio', 'Bp-section',
                'Bp-routingslip', 'Bp-signaturecomments', 'Bp-
signaturecontrol',
                'Bp-signaturetopaz', 'Bp-commentlog', 'Bp-tab-
strip',
                'Bp-tabstripend','Bp-tabcontent', 'Bp-tab-
contentend',
                'Bp-sectionembedded', 'Bp-sectionembeddedend',
'Bp-arraystart',
                'Bp-arrayend', 'Bp-removerowinarray', 'Bp-
arraymoveup',
                'Bp-arraymovedown', 'Bp-removerow', 'Bp-
addrow', 'Bp-sort', 'Bp-sum',
                'Bp-button', 'Bp-buttonarea', 'Bp-print-
button', 'Bp-savebutton',
                'Bp-cancelprocessbutton', 'Bp-showattach',
'Bp-userpicker',
                'Bp-grouppicker', 'Bp-contentpicker', 'Bp-con-
trolpicker', 'Bp-listbox',
                'Bp-slider', 'Bp-rating','Bp-lockform', 'Bp-
calculation',
                'Bp-datedifference', 'Bp-invite', 'Bp-sched-
uler', 'Bp-html',
                'Bp-datasourcepicker', 'Bp-categorypicker',
'Bp-attributepicker',
                'Bp-radiobuttonlist', 'Bp-formerrorstrings',
'Bp-forminfostrings',
                'Bp-kview', 'Bp-report', 'Bp-label', 'Bp-hot-
link', 'Bp-audit',
                'Bp-image', 'Bp-manageusers', 'Bp-emaildata',
'Bp-icon',
                'Bp-emailcompletelink', 'Bp-comment', 'Bp-com-
mentend', 'Bp-switch',
                'Bp-controlpicker', 'Bp-attachkview', 'Bp-
showattachkview',
                'Bp-arrayrownumber', 'Bp-sysvarform', 'Bp-
```

```
sysvartaskinstructions',
                'Bp-sysvarcurrentdate', 'Bp-sys-
varcurrentuser', 'Bp-sysvarstring',
                'Bp-sysvarcontrol', 'Bp-reauth', 'Bp-tooltip',
'Bp-activitylog',
                'Bp-captcha'
            ]
        },
        // These are the BP Logix control tool dropdown menus
        { name: 'bptoolbars', items: ['BPINPUT',
'BPOTHERINPUT', 'BPACTIONS',
            'BPOTHER', 'BPLAYOUT', 'BPRESPONSIVELAYOUT',
'BPARRAYS', 'BPATTACHMENTS',] },

//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!-
!!!!
    ];
```

ⓘ **In the CKEditor JavaScript, vertical divider lines between tools on the same toolbar are created with'-', while creating a new toolbar line is denoted with'/'. These conventions enable you to more easily organize the toolbars, and distinguish between different types of tool on each toolbar.**

Were you to place this configuration file in your installation as `/custom/ui/cust_ofd_config.js` and reference it via the `FormEditorConfig` Custom Variable, it would simply replicate the existing default configuration of the OFD toolbars. You wouldn't see a change, since this configuration is exactly the same as the Process Director default.

Using this configuration file, however, you could change the order in which toolbars and buttons appear, or remove them completely from displaying in your installation.

You should *ONLY* edit the CKEditor tools (Basically, just the tools highlighted with the bright blue code options in the sample above), and not the Process Director control tools, since that would disable them in your installation. BP Logix recommends that you don't edit any of the sections that are marked off by exclamation points at the start and end of the section. Technically, you could edit these sections to alter the order in which

tools or menus appear in the user interface of the Online Form Designer, but we strongly discourage it.

Customizing the Input Control Text Editor

The default configuration for the Input control's Text Editor would look like this:

```
CKEDITOR.editorConfig = function (config) {
    config.toolbar = [
        { name: 'document', items: ['Source', '-'] },
        { name: 'insert', items: ['Table', '-'] },
        { name: 'paragraph', items: ['NumberedList', 'Bul-
letedList', '-', 'JustifyLeft',
        'JustifyCenter', 'JustifyRight', 'JustifyBlock'] },
        '/',
        { name: 'basicstyles', items: ['Bold', 'Italic',
'Underline', '-'] },
        { name: 'colors', items: ['TextColor', 'BGColor', '-']
},
        { name: 'styles', items: ['Format', 'Font',
'FontSize'] },
    ]
}
```

Again, the `config.toolbar` command specifies the available tools in the Text Editor, and were you to place this file in your installation and reference it via the `sCkEditorCustomConfig` Custom Variable, you'd see no change in the UI.

You could modify this file to add or remove tools or toolbars, just as you do the OFD's configuration file. And, once again, you could simply use the `Config.removeButtons` command to remove unwanted buttons, Of course, since this is already a fairly minimal set of editing tools, your most likely use case will be to add tools to the existing ones, not remove the few that are there. To add new tools, you'll need to modify the configuration file to add the desired tools in the desired locations.

Adding Custom Fonts

While the default fonts provided with CKEditor are probably adequate for must purposes, your organization may wish to add custom fonts, for various reasons, such as using a specific corporate font, or barcode fonts for barcode readers. In this section, we'll refer to CKEditor generically, but the techniques we'll discuss apply equally to the Online Form Designer or Input control Text Editor.

Just as with customizing the toolbars for CKEditor, you'll need to add some new lines to the JavaScript configuration file. We'll also need to use the sUseCSS Custom Variable in the custom variables file.

This font functionality is completely separate from the toolbar configuration. You can customize just the fonts, customize both fonts and toolbars, or, of course, do neither.

Adding the Fonts

Our first step is to supply the font (or fonts, in this case) we'll want to use for customization. Let's say that we want to add a font named Nunito Sans and a barcode font named LibreBarcode39 to Process Director. The first thing we need to do is upload the **TrueType font files** for those fonts into the `/custom/ui` folder we've already been using. In this example, we'll upload **NunitoSans-Regular.ttf** and **LibreBarcode39Text-Regular.ttf** into the folder. These fonts will need to be accessible on the server so they can be shown to users that don't have them installed on their system.

We'll need to upload a custom CSS file (named "custom.css" in this example) into the `/custom/ui` folder. This CSS file will have the contents shown below:

```css
@font-face {
    font-family: "Nunito Sans";
    src: url(/custom/ui/NunitoSans-Regular.ttf);
}

@font-face {
    font-family: "Barcode";
    src: url(/custom/ui/LibreBarcode39Text-Regular.ttf);
}

.cke_editable {
    font: normal .9em "Nunito Sans";
}

.bpFormBody {
    font: normal .9em "Nunito Sans"!important;
}

.bpFormBody .gbleft {
    font-size: 17px;
}
```

This simple CSS does four things:

1. The two lines that start with the `@font-face` directive identify the two fonts we're adding and their location, so that users who do not have them installed can download them directly into their browser cache automatically, and see them on the screen.

2. The `.cke_editable` class is the default text display class for CKEditor. This CSS will override the default text style and replace it with Nunito Sans as the new default font to use in CKEditor.

3. The `.bpFormBody` class is a built-in display class that sets the default font for Form text when viewed in running Forms. By default, this font is set to Arial, and we're going to override it with Nunito Sans. Because we're overriding the BP Logix style sheet, we need to add the `!Important` directive to make this work. If we don't add this class, all of our default text styling will show up as Nunito Sans in CKEditor, but end users will see Arial as the default font when they view the Forms. (Technically, we could skip this CSS class, but then, every time we created a Form, we'd have to specifically style all our text using the Font tool in CKEditor. Changing the default font is just...lots easier.)

4. Finally, the `.bpFormBody .gbleft` class will ensure that Form buttons, such as the Button Area control that displays the Submit, Cancel, activity result buttons, etc., will display the button icons properly. We're changing the default font on Forms, which will affect how Form buttons display, since Nunito Sans has a different line height than the default Arial font. Setting the font size to 17px tweaks the spacing just enough to make button icons appear vertically centered in Form buttons.

   Our problem is that we can't just edit the `.gbleft` class by itself. It's a universal class that affects all buttons in Process Director. That includes buttons that appear in the UI itself. We don't want to throw off the placement of button icons or images used in the main product UI.

   All we want to do is ensure that buttons display icons properly on Forms. So, we need to use the syntax `.bpFormBody .gbleft`, with a space between the two classes. In CSS, this tells the browser that we only want to make this change to the `.gbleft` class if it's displayed inside an element that's styled with the `.bpFormBody` class. Since only the `<body>` element of a running Form instance uses the `.bpFormBody` class, and Form buttons are always displayed inside the `<body>` element, this style will only affect Form buttons on Running Forms. All of the other buttons used elsewhere in the product won't

be affected.

Also, we don't need to use the `!important` directive, since this syntax essentially creates a unique class that automatically overwrites the default `.gbleft` class on running Forms, due to it's position in the CSS hierarchy.

> ⓘ **Process Director's CSS styling is very complex. You should be aware that creating custom styles will be time-consuming, and will require considerable knowledge of CSS to pull off correctly, without adversely affecting the UI of the product itself.**

Editing the Configuration file

Next we'll need to edit our custom configuration file(s) for CK editor. We need to add two lines of configuration:

```
CKEDITOR.editorConfig = function (config) {
    // New fonts to add to the system
    config.contentsCss = "/custom/ui/custom.css";
    config.font_names = 'Nunito Sans;Barcode;' + config.font_names;

    // All the toolbar config stuff we added previously,
    // Which is completely separate from the font stuff.
config.toolbar = [
        ... // Code for Toolbar configuration
    ]
}
```

The `config.contentsCss` command let's CKEditor know that we want to apply some custom CSS to CKEditor, and specifies the relative URL of our custom CSS file.

The `config.font_names` command adds our two new fonts to the list of fonts that are already shown in CKEditor's Font tool. This particular syntax will add our fonts to the top of the CKEditor Font tool's list of existing fonts, with all of the standard fonts appearing below them in the Font tool.

Our final step is to add our custom CSS file to our existing installation's configuration, as described in the Adding a Custom CSS Stylesheet section, above.

> ⓘ **If you want the new fonts used as the default font for both the OFD and the Input control, you'll need to add these lines to *both* configuration files.**

Once you've completed these steps, the two new fonts will be available for use in CKEditor, and will display properly in Process Director for end users.

Here's how it looks in the OFD:
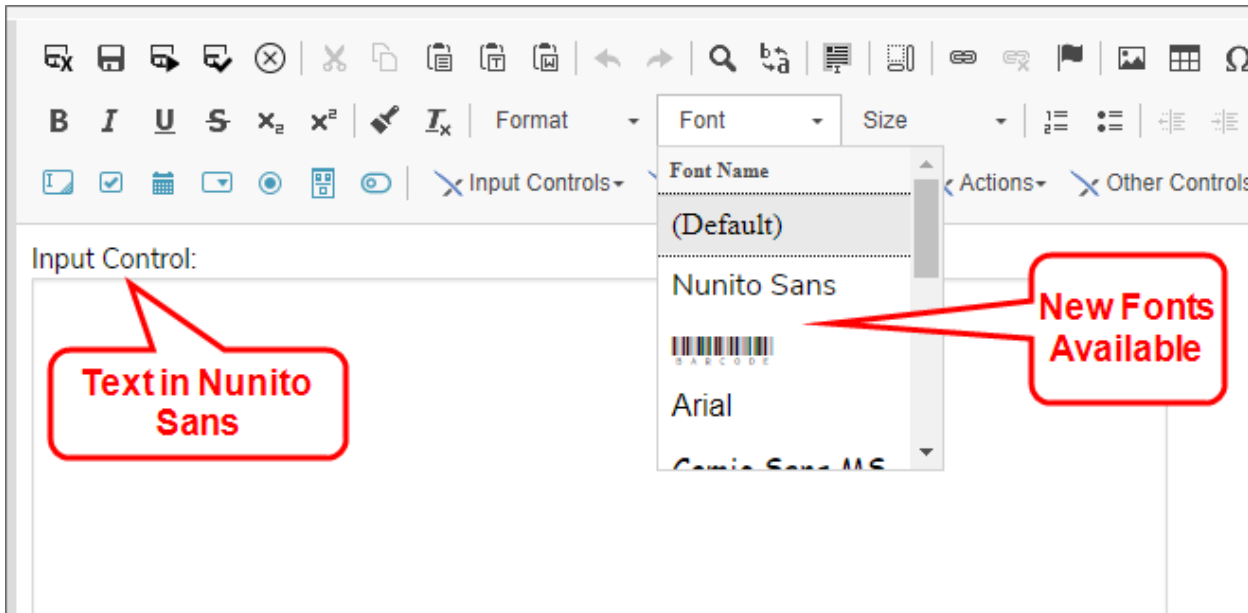
And here's how it looks in a running Form that uses the Input control's Text Editor:
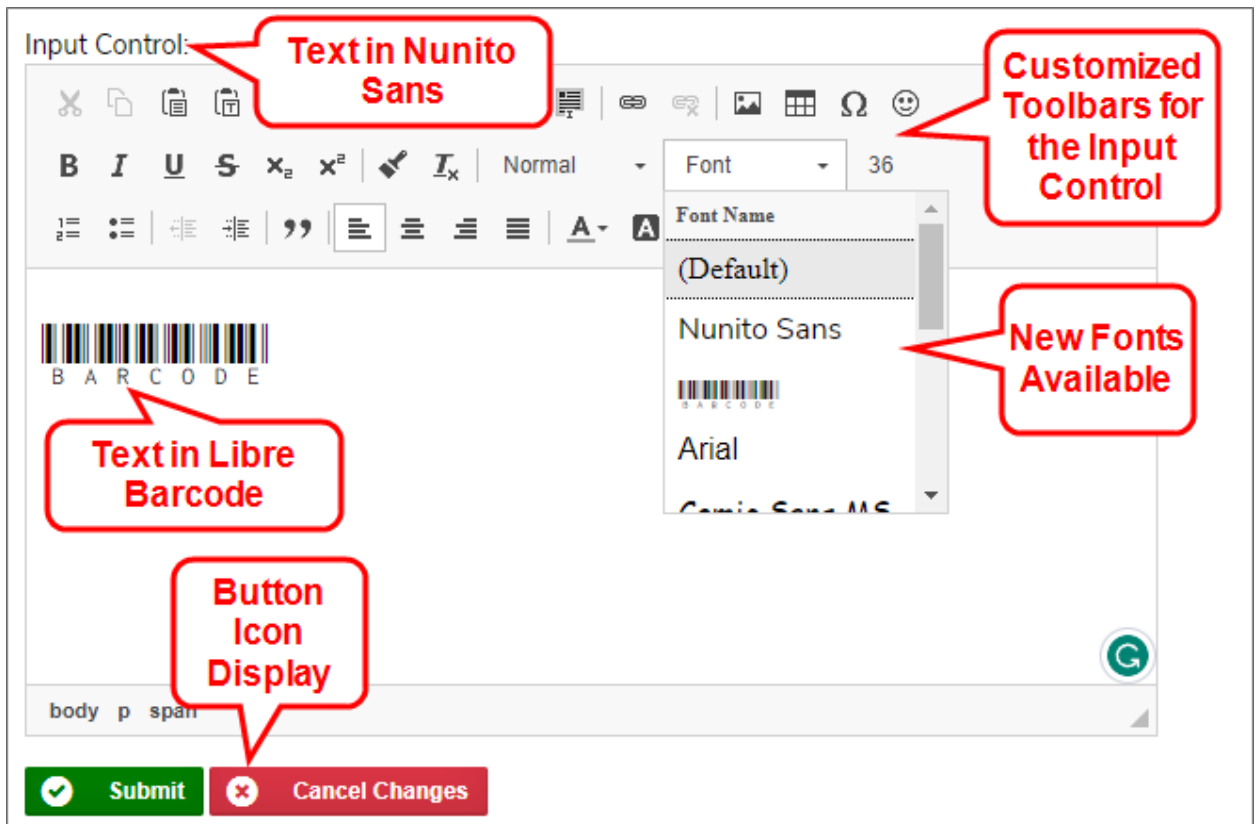
Environment Message Customization

The Server Control page of the IT Admin area's Troubleshooting section enables you to display an environmental message, e.g., "Staging Server" at the top of every page displayed to user, in order to let them know what environment they're using. This message can be styled in a custom CSS stylesheet, by setting the desired properties for the `.bpCustomInfoText` style. The most common customization would be to change the background color of the message, using the style syntax:

```
.bpCustomInfoText {
    background-color: #FFE600;
}
```

For this style to appear to users, you must add a custom CSS stylesheet to Process Director, as described above, if you do not already have one configured.

Using Web Services

This section provides a reference for calling or extending Web Services in Process Director. Documentation for each web service is located in the Available Web Services topic.

You'll need to enable the Web Services in BP Logix using the Properties page of the IT Admin area's Installation Settings section. Ensure that Web Service Enabled is set to "True". You can optionally require all Web Service calls to authenticate by setting Require Web Service Authentication to "True". When this is set, you'll need to call the **Authenticate** Web Service call prior to any other calls. You can optionally set the Web Service Restrictions property to a list of comma separated IP addresses that can call Web Services. If this field is set, only requests from these IP Addresses will be allowed.

Web service calls can also be made via SQL Server database triggers. The following query in SQL is an example of how to create a trigger that will make a web service call in Process Director.

```
USE [DatabaseName]
    GO
        SET ANSI_NULLS ON
    GO
        SET QUOTED_IDENTIFIER ON
    GO
        ALTER TRIGGER [dbo].[Trigger_TableName]
            ON [dbo].[TableName]
        AFTER INSERT
        AS
            BEGIN
                SET NOCOUNT ON;
                DECLARE @vPointer INT
                EXEC sp_OACreate 'MSXML2.ServerXMLHTTP',
@vPointer OUTPUT
                EXEC sp_OAMethod @vPointer, 'open', NULL,
'GET', 'http://servername/
                    /Services/wsWorkflow.asmx/Run?WFID=wfid'
                EXEC sp_OAMethod @vPointer, 'send'
                EXEC sp_OADestroy @vPointer
            END
```

For confirmation of the web services call, you can declare some additional fields if you want and change the last few lines to:

```
EXEC sp_OAMethod @vPointer, 'responseText'
EXEC sp_OAMethod @vPointer, 'Status', @vStatus OUTPUT
EXEC sp_OAMethod @vPointer, 'StatusText', @vStatusText OUTPUT
EXEC sp_OADestroy @vPointer
SELECT @vStatus AS Status, @vStatusText AS StatusText,
    @vResponseText AS Response
```

REST Services

Microsoft natively provides a non-SOAP (REST) interface to some web services, if the service requires no parameters, or only parameters of the primitive types bool, int, or string. To check to see if a particular web service supports a REST interface, you can navigate to the service page for a web service, e.g., the "services/wsForm.asmx" page, and click on one of the web service calls. If the web service call provides an "HTTP GET" operation, then it can use a REST call to run the service. *This isn't a feature that is managed or edited by BP Logix; it is a native feature of web services that has been implemented by Microsoft, and is subject to change at their discretion.*

You can call these services via with the standard HTTP GET protocol (using a simple URL). A custom variable, fWebServiceAllowCredentialsURL is set to "true" as a default, to allow credentials to be passed on the URL.

REST URLs should be in the following format:

```
http://RESTSer-
viceURL?parameterName=parameterValue&otherParam=otherValue
```

The "bpUserId" and "bpPassword" parameters are necessary to authenticate with Process Director when using REST APIs. The credentials you pass can be for any user that has permission to invoke a Web Service, irrespective of whether that user is a System Administrator.

> ⓘ **Be advised that using the bpUSERID or bpPassword parameter requires sending the User ID and Password in clear text, so be mindful of the security implications of transmitting these values and limit the calls to the localhost. Secure the rest calls using the IP address white list in the installation settings.**

Other REST Services

In addition to the built-in REST services in Process Director, implementers can also use Business Values to retrieve and use REST data in any desired context. Additionally, two Web Service Custom Tasks to retrieve REST data from any accessible REST web service to Fill Fields or Fill Dropdowns with REST data.

For more information about accessing and using external REST services, please refer to the REST Services topic.

Web Service Authentication Settings

A web service call can be made without requiring authentication if the web service is being called from a source that is listed as a "Local IP Address" in the installation settings in Process Director.

If an IP address is listed in the "Web Service Restrictions" than web service calls can only be made from those IP addresses.

Server Name	BP Logix Process Director
Interface URL	
Proxy URL	
Store Documents On File System	True
Document Storage Path	C:\DemoDocStorage
Document Size For File System Storage	105000000
Local IP Addresses	
Load Balanced URLs	
Registered Email	
Logging Level	2
CSS	
Logo URL	
Logo Link	
Home Page Top Height	0
Home Page Logo Width	0
SMTP Host	
SMTP Port	0
SMTP UserID	
SMTP Password	••••••••••
SSL Required?	False
Web Service Enabled	True
Require Web Service Authentication	False
Web Service Restrictions	
Number of Custom Icons	35

Extending BP Logix Web Services

You can also write new or extend the web services provided by BP Logix by developing your own custom .ASMX pages in the custom folder. Your new Web Service, for example, can be a "proxy" to a remote web service that takes a parameter list that the Web Service Custom Tasks support. Or you can provide Web Services that search for and return custom data.

See the sample in the /custom/samples/SampleService.asmx.sample file. Notice that the Web Service is derived from the bpWebService class. This enables you to call any BP Logix SDK API from the new Web Service.

To extend BP Logix Web Services inside Visual Studio, use the fully functional Visual Studio project installed with the product named bpVS.zip. Refer to the sample file SampleService.asmx.

Calling Other Web Services

The easiest way to call Web Services provided by other Enterprise Applications is to use the Web Service Custom Task. This Custom Task enables you to call a remote Web Service without writing any code. You can map the inputs and outputs of the Web Service call to Form Fields, system variables, or custom variables.

Additionally, you can use any .NET language to call other Web Services. To do this, use the .NET WSDL compiler to generate the proxy code for the Web Service. Then package the proxy into a .DLL and place it into the /Bin folder of the BP Logix application. Your custom Form, Workflow, or Knowledge View scripts can then call these Web Services.

Available Web Services

There are many APIs that can be called from any platform that supports Web Services. The BP Logix Web Services are divided into functional areas, providing separate WSDLs for each area.

The table below briefly describes the Web Service areas. The name of each web service is linked to the topic page containing the documentation for all of the Web Service calls available for that web service.

WEB SERVICE	DESCRIPTION	DOCUMENTATION URL / WSDL URL
wsAdmin	Services to retrieve administrative information, authenticate,	`http://<servername>/Services/wsAdmin.asmx` `http://<servername>/Services/ wsAdmin.asmx?WSDL`

WEB SERVICE	DESCRIPTION	DOCUMENTATION URL / WSDL URL
	and set context.	
wsCase	Services to manipulate Case instances and data.	`http://<servername>/Services/wsCase.asmx` `http://<servername>/Services/ wsCase.asmx?WSDL`
wsContent	Services to manipulate content items (get, delete, get/-set categories and meta data, etc), in the BP Logix repository.	`http://<servername>/Services/wsContent.asmx` `http://<servername>/Services/wsContent.asmx?WSDL`
wsForm	Services to manipulate Forms including getting and setting form data.	`http://<servername>/Services/wsForm.asmx` `http://<servername>/Services/wsForm.asmx?WSDL`
wsGroup	Services to iterate and manipulate groups.	`http://<servername>/Services/wsGroup.asmx` `http://<servername>/Services/wsGroup.asmx?WSDL`
wsReport	Services to run a report	`http://<servername>/Services/wsReport.asmx` `http://<servername>/Services/wsReport.asmx?WSDL`

WEB SERVICE	DESCRIPTION	DOCUMENTATION URL / WSDL URL
wsRule	Services to manipulate Business Rules	`http://<servername>/Services/wsRule.asmx` `http://<server-name>/Services/wsRule.asmx?WSDL`
wsTimeline	Services to query, start, and manipulate Process Timelines.	`http://<server-name>/Services/wsTimeline.asmx` `http://<server-name>/Services/wsTimeline.asmx?WSDL`
wsUser	Services to iterate and manipulate users.	`http://<servername>/Services/wsUser.asmx` `http://<server-name>/Services/wsUser.asmx?WSDL`
wsUtil	Utility Services such as version query and a user authenticate.	`http://<servername>/Services/wsUtil.asmx` `http://<server-name>/Services/wsUtil.asmx?WSDL`
wsWork-flow	Services to query, start, and manipulate Workflows.	`http://<server-name>/Services/wsWorkflow.asmx` `http://<server-name>/Services/wsWorkflow.asmx?WSDL`

Where `<servername>` is the host where the product is installed.

Your specific development environment will have documentation to process a WSDL files and calling web services.

Windows Communication Foundation (WCF) can also be used to call any web service in Process Director. All Process Director web services have a Web Services

Description Language (WSDL) interface, and any WCF client can consume the web service. Using WCF also allows you to call web services asynchronously.

Service Handle Method

For Process Director v6.1.300, The ServiceHandle method for web services was added to the SDK, under the `BPLogix.WorkflowDirector.SDK` namespace, to enable the creation of custom web services.

This method can be called to perform some complex functions, such as exporting users from an on-premise Active Directory system to a Cloud installation of Process Director. The method can be called with the declaration synatax:

```
public ServiceHandle(bpWebServiceInternal pService,
                     string pAPI,
string method,
                     string sourceFile,
                     int line,
params object[] pParams)
```

Most of the parameters are not enforced if they use the sub-method `ServiceHandle.Create` instead of creating the ServiceHandle object manually.

This method is normally implemented as a web service via a custom ASCX control.

wsAdmin

The following methods are available in the wsAdmin service.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: Indication of success.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

GetDatabaseInfo

This method will return various information about Process Director's database.

Input Parameters

Information: The specific information that should be returned. This parameter will accept any of the following values:

- getMaxDBSize
- getMaxSizeNoLog
- getDBSize
- getLogSize
- getDBSizeNoLog
- getMaxLogSize
- getFreeSpace

Returns

Size: Size of the database in megabytes

GetDisabledUsers

This method will get the number of disabled users in Process Director's database.

Input Parameters

None.

Returns

disabledUsers: The number of disabled users in the database.

GetDiskInfo

This method will return info on all local drives.

Input Parameters

None.

Returns

allDrives: The Drive Information for each local drive.

GetLoggedInUsers

This method will get the number of users currently logged in to Process Director.

Input Parameters

None.

Returns

loggedIn: The number of users logged in.

GetMaximumUsers

This method will get the maximum possible number of users for Process Director's license.

Input Parameters

None.

Returns

possibleUsers: The number of possible users.

GetObjectsByType

This method will get the number of specific types of objects in Process Director's database.

Input Parameters

type: The type of object to be returned. This parameter accepts one of the following:

- Workflow
- timeline
- Form
- folder
- document

Returns

objects: number of objects in the database.

GetServerInfo

This method will return information about the server.

Input Parameters

Information: The specific server information to be returned by the web service. The parameter will accept one of the following:

- CurrentMemory
- PeakVirtual
- CurrentVirtual
- PeakMemory
- ProcdessorTime
- StartTime
- PDVersion

Returns

ServerInfo: Requested information about the server as specified in the input parameter.

GetTotalActiveUsers

This method will return the number of active users in Process Director's database.

Input parameters

None.

Returns

activeUsers: number of active users in the database.

GetTotalGroups

This method will get the total number of groups in Process Director's database.

Input parameters

None.

Returns

groups: The number of groups in the database.

GetTotalObjects

This method will get the number of objects in Process Director's database.

Input parameters

None.

Returns

objects: number of objects in the database.

GetTotalUsers

This method will get the total number of users in Process Director's database.

Input parameters

None.

Returns

Users: The number of users in the database.

SendMessageToAll

This method will send a message to all users on a server.

Input parameters

message: The message to be sent to all users.

Returns

success: A Boolean value reflecting success of web services (FALSE if errors, TRUE if successful).

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean:indication of success.

wsCase

The following methods are available in the wsAdmin service. This service is available to users of Process Director v4.02 or higher.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: The indication of success.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

CreateCase

This method will create a new case instance and pass in initial case property values.

Input Parameters

PID: The type of object to be returned.

PathName: The full path of the case definition or a CASEID.

CaseValues: The list of name/value pairs representing the case properties and their values to be set, passed as a List<NameValue> object.

Returns

The Case instance of the newly instantiated case, or null if there is an error.

CreateSimpleCase

This method will create a new case instance. Initial case property values can be passed in through the query string.

Input Parameters

PID: The type of object to be returned.

PathName: The full path of the case definition or a CASEID.

Returns

The Case instance of the newly instantiated case, or null if there is an error.

GetCaseByCASEINSTID

This method will return a Case instance from it's Case Instance ID.

Input Parameters

CASEINSTID: The ID of the Case Instance

Returns

The specified Case Instance, or null if not found.

GetCaseData

This method will return a list of all the properties in a case instance and their associated values.

Input Parameters

CASEINSTID: The ID of the case instance.

Returns

A List<NameValue> object containing a list of the case properties and their values.

GetCaseProperties

This method will return a list of the properties associated with a case definition.

Input Parameters

CASEID: The ID of the case Definition.

Returns

A List<NameValue> object containing a list of the case properties and their types.

Instantiate

This method will create a new case instance.

Input Parameters

CASEID: The ID of the case definition to instantiate

Returns

The Case instance of the newly instantiated case.

SetCaseData

This method will set the value of all specified properties in a case instance.

Input Parameters

CASEINSTID: The ID of the case instance.

CaseData: A list of the name/value pairs containing the names of the properties and their respective values, passed as a List<NameValue> object.

Returns

Boolean: True if the operation succeeds.

SetCaseProperty

This method will set the value of a specified property for a case instance.

Input Parameters

CASEINSTID: The ID of the case instance.

PropertyName: The name of the property whose value will be set.

Value: The value to which the property will be set.

Returns

Boolean: True if the operation succeeds.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls

Returns

Boolean: True if the operation succeeds.

wsContent

These web services allow you to manipulate Process Director content objects pro-grammatically.

AppendPath

This method will create all subfolders under the indicated folder by the path to append.

Input parameters

pFID: The ID of the Folder.

pPath: The path to append underneath the folder.

Returns

Folder: The new folder corresponding to the path we appended.

AttributeExists

This method returns a boolean value after determining whether a specified Meta Data Attribute exists. If the Attribute exists, the service will return "true".

Input parameters

PID: The Partition ID or Name containing the folder.

CategoryName: The name of the category to evaluate.

Returns

Boolean: Whether the Attribute exists.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: The indication of success.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

CategoryExists

This method returns a boolean value after determining whether a specified Meta Data Category exists. If the Attribute exists, the service will return "true".

Input parameters

PID: The Partition ID or Name containing the folder.

AttributePath: The name of the category to evaluate.

Returns

Boolean: Whether the Category exists.

CreateNewFolder

This method will create all subfolders under the indicated folder by the path to append.

Input parameters

pFID: The ID of the Folder

pPath: The path to append underneath the folder.

Returns

Folder: The new folder corresponding to the path you appended.

CreatePath

This method will create a path including all subfolders under the indicated partition.

Input parameters

pPID: The ID of the Partition.

pPath: The path to create in the Partition.

Returns

Folder: The new folder corresponding to the path you created.

DeleteObject

This method deletes an object from the content repository.

Input parameters

ID: The ID of the object.

Returns

None.

DeleteObjectAndChildren

This method deletes an object and ALL children and sub-Workflows (if this is a process instance) from the content repository.

Input parameters

ID: The ID of the object.

Returns

None.

GetAttribute

This method will get an object's attribute value.

Input Parameters

ID: The ID of the object.

Category: The name of the category.

Attribute: The name of the attribute.

Returns

The attribute value

GetFolderByID

This method will get a folder by its ID.

Input Parameters

ID: The ID of the Folder.

Returns

Folder: The actual folder or null if not found.

GetFolderByPathName

This method will get a folder from its path in the repository.

Input parameters

PID: The Partition ID or Name containing the folder.

PathName: The full path of the folder (e.g. /My Documents/sales/).

Returns

Folder: The actual folder or null if not found.

GetObjectByID

This method will get an object from its ID.

Input parameters

ID: The ID of the object.

Returns

ContentObject: The actual object or null if not found.

GetObjectByPathName

This method will get an object from its path in the repository.

Input parameters

PID: The Partition ID or Name containing the object.

PathName: The full path of the object (e.g. /My Documents/Document.doc).

Returns

ContentObject: The actual object or null if not found.

GetObjectsFromParentID

This method will return all objects that are children of the object at the specified path.

Input parameters

PID: The Partition ID or Name containing the object.

Path: The full path of the object (e.g. /My Documents/Document.doc).

Returns

List: A list containing the children of the parent object.

GetObjectsFromParent

This method will return all objects that are children of the object at the specified path.

Input parameters

PID: The Partition ID or Name containing the object.

Path: The full path of the object (e.g. /My Documents/Document.doc).

Returns

List: A list containing the children of the parent object.

GetPartitions

This method will return all objects that are children of the object at the specified path.

Input parameters

None

Returns

List: A list containing the Partition attributes.

GetRootFolder

This method will get the root folder object for a partition.

Input parameters

PID: The Partition ID or Name to find

Returns

FolderObject: The actual root object or null if not found.

ImportXML

This method imports an XML package into the Content List.

Input parameters

PID: The Partition ID where the import will occur

ParentFolderID: The ID parent object for the import

XMLData: The bytes of the XML document to import

Returns

RetMsg: The list of output strings after the import

MoveObject

This method moves a content object.

Input parameters

ID: The ID of the object to be moved.

DestinationID: The object under which the item identified by the ID will be moved.

Returns

None.

RunKView

This runs a Knowledge View and returns the results.

Input parameters

KVID: The Knowledge View ID to run.

Filter: The optional list of filter data.

StartFID: The optional Folder ID to start.

StartCATID: The optional Category ID to start.

Returns

Output: The list of output rows.

SetAttribute

This method will set an object's attribute value (and indirectly its category).

Input parameters

ID: The ID of the object.

Category: The name of the category.

Attribute: The name of the attribute.

Value: The value to set.

Returns

None.

SetAttributes

This method will set an object's attribute values (and indirectly its category).

Input parameters

ID: The ID of the object.

Attributes: The List of attribute names and values.

Returns

None.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean: The indication of success.

SetExternalAttribute

This method will set an object's attribute based on its external attribute.

Input parameters

ID: The ID of the object.

Name: The name of the external attribute.

Value: The value to set.

Returns

None.

SetExternalAttributes

This method will set an object's attributes based on its external attribute.

Input parameters

ID: The ID of the object

Attributes: The List of External attribute names and values

Returns

None.

SetGroupName

This method will set an object's Group Name property.

Input parameters

ObjectID: The ID of the object

ParentID: The ID of the object's parent object

GroupName: Group name to be assigned

Returns

None.

SetMetaData

This method will get an object's meta data and optionally attribute values.

Input parameters

ID: The ID of the object

XML_Data: The XML string representing the Category and Attribute values to set

Returns

None.

wsForm

These web services enable you to manipulate Process Director forms pro-grammatically.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True of the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

CreateForm

This method will create a new form instance and pass in initial form values.

Input Parameters

PID: The Partition ID or Name.

PathName: The full path of the form definition (e.g. /Forms/My Form) or a FORMID.

FormValues: The list of name/value pairs to set in the new form instance.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

Returns

FormInstance: The new form instance or null if an error prevents execution of the operation.

CreateFormEx

This method will create a new form instance and pass in initial form values. Unlike CreateForm, CreateFormEx allows you to pass in the name of the new form

instance manually.

Input Parameters

PID: The Partition ID or Name.

PathName: The full path of the form definition (e.g. /Forms/My Form) or a FORMID.

FormValues: The list of name/value pairs to set in the new form instance.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

FormInstanceName: Optionally used to set new form instance name.

Returns

FormInstance: The new form instance or null if an error prevents execution of the operation.

CreateFormEx2

This method will create a new form instance and pass in initial form values, and will include both the Value and Display Text for dropdown fields. CreateFormEx2 has the same options as CreateFormEx, but instead of taking a List<NameValue> parameter to set the form field values, it takes a List<NameTextValue> parameter. The NameValue type only allows you to set a name, value, and number (which is simply a numerical representation of the field's value). The NameTextValue also allows you to set a Text property, which can be distinct from the Value property. This is particularly useful when setting dropdowns, where the text of a dropdown selection and its value are often different.

Input Parameters

PID: The Partition ID or Name.

PathName: The full path of the form definition (e.g. /Forms/My Form) or a FORMID.

FormValues: The list of name/value pairs to set in the new form instance.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

FormInstanceName: Optionally used to set new form instance name.

Returns

FormInstance: The new form instance or null if an error prevents execution of the operation.

CreateSimpleForm

This method will create a new form instance. Initial form values can be passed on the query string, which enables you to create a form and populate form fields. Form fields are identified using a URL parameter with the prefix "EXT_". For instance, a Field called "FirstName" on the Form can be populated via URL Parameter as "EXT_FirstName=Bob". This would result in the following URL:

```
http://server-
name.com/services/wsform.asmx/CreateSimpleForm?PID=ID&
    PathName=Path&SkipDefaultValues=false&EXT_FirstName=Bob
```

Input Parameters

PID: The Partition ID or Name.

PathName: The full path of the form definition (e.g. /Forms/My Form) or a FORMID.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

EXT_SomeFieldName: A form field you wish to populate with a value when the Form is created.

Returns

FormInstance: The new form instance or null if an error prevents execution of the operation.

GetFormByFORMINSTID

This method will get a form instance from its FORMINSTID.

Input Parameters

FORMINSTID: The ID of the Form Instance.

Returns

FormInstance: The actual form instance or null if not found.

GetFormData

This method will get all form data from a form instance.

Input Parameters

FORMINSTID: The ID of the Form Instance.

Returns

List: The list of name/value pairs in this form instance.

GetFormDataEx

This method will get all form data from a form instance, and will include both the Value and Display Text for dropdown fields.

Input Parameters

FORMINSTID: The ID of the Form Instance.

Returns

List: The list of name/value pairs in this form instance.

GetFormSchema

This method will return the form schema for all form controls.

Input Parameters

FORMID: The ID of the Form Definition.

Returns

List: The list of form fields and their types.

GetFormSchemaEx

This method will return the form schema for all form controls, and will include both the Value and Display Text for dropdown fields.

Input Parameters

FORMID: The ID of the Form Definition.

Returns

List: The list of form fields and their types.

Instantiate

This method will instantiate a new form instance.

Input Parameters

FORMID: The ID of the Form to create.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

Returns

FormInstance: The new form instance or null if an error prevents execution of the operation.

RecalcFormInstanceName

This method will force the form instance name to be recalculated.

Input Parameters

FORMINSTID: The ID of the Form Instance.

Returns

Boolean: True if the operation succeeds.

SearchForms

This method will search form instances for values.

Input Parameters

PID: The Partition ID or Name.

FORMID: Optional FORM ID to limit search.

ControlName: The name of the form control to search.

Value: The value to search for.

Returns

SearchResult: An array of search results that match the search criteria. The array contains the following values for each form that is returned in the array.

- PID
- FORMID
- FORMINSTID
- FCID

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean: The indication of success.

SetFormData

This method will set form data for a form instance.

Input Parameters

FORMINSTID: The ID of the Form Instance.

FormValues: The list of name/value pairs to set in this form instance.

Returns

Boolean: True if the operation succeeds.

SetFormDataEx

This method will set form data for a form instance, and will include both the Value and Display Text for dropdown fields.

Input Parameters

FORMINSTID: The ID of the Form Instance.

FormValues: The list of name/value pairs to set in this form instance.

Returns

Boolean: True if the operation succeeds.

SetFormDataField

This method will set form data for a single form field for a form instance.

Input Parameters

FORMINSTID: The ID of the Form Instance.

FieldName: The form field name to set.

FieldName: The value of the form field.

Returns

Boolean: True if the operation succeeds.

wsGroup

These web services enable you to manipulate Process Director user groups pro-grammatically.

AddGroupToWorkspace

This method will add a Group to a specified Workspace.

Input Parameters

GID: The Group ID of the Group to add.

PROFILEID: The ProfileID of the Workspace to which to add the user.

Returns

Boolean: True if the operation succeeds.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

CreateGroup

This method will create a group.

Input parameters

GroupName: The Group Name to add.

Returns

Group: The new group object that was created, or null if an error prevents execution of the operation.

DeleteGroup

This method will delete a group.

Input parameters

GID: The GID of the group to delete

Returns

Boolean: True if the operation succeeds.

EnumerateUserGroups

This method returns a list of all groups in the system.

Input parameters

None

Returns

String: Comma-separated list of groups.

GetGroupByID

This method will get a Group from its ID.

Input parameters

GID: The ID of the group to retrieve.

Returns

Group: The actual Group or null if the Group isn't found.

GetGroupByName

This method will get a Group from its Name.

GroupName: The name of the group to retrieve.

Returns

Group: The actual Group or null if the Group isn't found.

GetUsers

This method will get a list of users assigned to a Group.

Input parameters

GID: The ID of the group to retrieve.

Return

User: An array containing all of the users assigned to the group. For each user, the array will contain the following values:

- UID
- AuthType
- UserName
- UserID
- Email
- AvgLoginSecs
- Groups: An array containing all of the groups of which the user is a member

  - Group

    - GID
    - GroupName

RemoveFromGroupWorkspace

This method will remove a Group from a specified Workspace.

Input Parameters

GID: The Group ID of the Group to remove.

PROFILEID: The ProfileID of the Workspace from which to remove the Group.

Returns

Boolean: True if the operation succeeds.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input parameters

UID: The ID of the user to use for the context of all calls

Returns

Boolean: True if the operation succeeds.

wsReport

These web services enable you to manipulate Process Director reports programmatically.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

ExportReport

This method will export a report to the local file system or into the Content List.

Input Parameters

RID: The Report ID of the report to export.

ExportName: The file name to export with extention (PDF, DOCX, XLSX, PPTX, etc.).

ContentParentID: The ID of the parent in the Content List to save the exported file.

ContentFolderPath: The full folder path in the Content List to save the exported file.

LocalFolderPath: The path to the folder in the local file system to save the exported file.

Returns

Boolean: True if the operation succeeds.

ExportReportEx

This method will export a report to the local file system or into the Content List.

Input Parameters

RID: The Report ID to export.

ExportName: The file name to export with extention (PDF, DOCX, XLSX, PPTX, etc.).

ContentParentID: The ID of the parent in the Content List to save the exported file.

ContentFolderPath: The full folder path in the Content List to save the exported file.

LocalFolderPath: The path to the folder in the local file system to save the exported file.

Variables: The list of variables to pass to the report.

Returns

Boolean: True if the operation succeeds.

GetReportByRID

This method will get a report definition from its RID.

Input Parameters

RID: The ID of the Report Definition

Returns

R**eportDefinition:** The actual report definition or null if the report isn't found.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls

Returns

Boolean: True if the operation succeeds.

wsRule

These web services enable you to manipulate Process Director Business Rules programmatically.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

Evaluate

This method will evaluate a rule.

Input Parameters

RULEID: The ID of the Rule Definition.

Context: Optional context passed to rule.

Variables: Optional list of variables passed to rule.

Returns

The result of the rule evaluation.

GetRuleByRULEID

This method will get a rule definition from its RULEID.

Input Parameters

RULEID: The ID of the Rule Definition.

Returns

RuleDefinition: The actual rule definition or null if not found.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean: True if the operation succeeds.

wsTimeline

These web services enable you to manipulate Process Timelines programmatically.

AddToTimeline

This method will add an object to a timeline instance.

Input Parameters

TLINSTID: The ID of the Timeline Instance.

ID: The ID of the object to add.

Type: ObjectType - The type of object to add.

Group: The optional group name to add the object into.

Returns

TimelineInstance: The timeline instance in which the call ran (or null if an error occurred).

AddUsersToActivity

This method will add user(s) to a specific Timeline instance.

Input Parameters

ACTINSTID: The Timeline Activity Instance ID

UID: Comma separated list of UIDs to add

Returns

Boolean: True if the operation succeeds.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation

token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

Cancel

This method will cancel a running Timeline instance.

Input Parameters

TLINSTID: The ID of the TimelineInstance

Returns

TimelineInstance: The actual Timeline instance that was canceled, or Null if not found.

GetActivityByACTID

This method will return a Timeline Activity for a specific Timeline instance.

Input Parameters

TLINSTID: The Timeline Instance ID.

ACTID: The Timeline Activity ID to retrieve.

Returns

TimelineActivity: The parent Timeline Activity for the timeline instance passed to the service (or null if an error occurred).

GetActivityByName

This method will return a Timeline Activity for a specific Timeline instance.

Input Parameters

TLINSTID: The Timeline Instance ID.

ActName: The Timeline Activity name to retrieve.

Returns

TimelineActivity: The parent Timeline Activity for the timeline instance passed to the service (or null if an error occurred).

GetTimelineByTLID

This method will get a Timeline definition from its TLID.

Input Parameters

TLID: The ID of the Timeline Definition.

Returns

TimelineDefinition: The actual Timeline definition or null if the timeline isn't found.

GetTimelineByTLINSTID

This method will get a timeline instance from its TLINSTID.

Input Parameters

TLINSTID: The ID of the Timeline Instance.

Returns

TimelineDefinition: The actual Timeline definition or null if the timeline isn't found.

Instantiate

This method will instantiate a Timeline Instance from a TLID.

Input Parameters

TLID: The ID of the Timeline Definition to instantiate.

InitiatorUID: The optional UID of the Timeline initiator.

Returns

TimelineInstance: The actual Timeline instance or null if the timeline isn't found.

PostEvent

This method will post an event to a timeline that is waiting on a Wait activity.

Input Parameters

TLINSTID: The optional ID of the Timeline Instance to POST

EventName: The optional Event Name to post.

Returns

Boolean: True if the operation succeeds.

RemoveUsersFromActivity

This method will remove user(s) from a specific activity instance.

Input Parameters

ACTINSTID: The Timeline Activity Instance ID.

UID: Comma separated list of UIDs to add.

Returns

Boolean: True if the operation succeeds.

Restart

This method will restart an existing Timeline instance TLINSTID at an optional activity ACTID.

Input Parameters

TLINSTID: The ID of the Timeline Instance to restart.

ACTID: The ACTIVITY ID of the ACTIVITY in the Timeline definition to restart (optional).

Returns

TimelineInstance: The Timeline instance which the call restarted (or null if an error occurred).

Run

This method will instantiate and run a Timeline Instance from a TLID.

Input Parameters

TLID: The ID of the Timeline Definition to run.

Returns

TimelineInstance: The Timeline instance which the call ran (or null if an error occurred).

RunTimeline

This method will create and run a new Timeline instance and optionally add a Timeline object.

Input Parameters

PID: The Partition ID or Name.

PathName: The full path of the Timeline definition (e.g. /Process/My Timeline).

ID: The ID of the object to add.

Type: ObjectType - The type of object to add.

Group: The optional group name to add the object into.

InitiatorUID: The optional UID of the Timeline initiator.

Returns

TimelineInstance: The Timeline instance which was created (or null if an error occurred).

RunTimelineWithForm

This method will create a new form instance, a new Timeline instance and attach the form.

Input Parameters

PID: The Partition ID or Name.

TimelinePathName: The full path of the Timeline definition (e.g. /Process/My Timeline).

FormPathName: The full path of the form definition (e.g. /Forms/My Form).

FormValues: The list of name/value pairs to set in the new form instance.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

Group: The optional group name to add the object into.

InitiatorUID: The optional UID of the Timeline initiator.

Returns

TimelineInstance: The Timeline instance which was created (or null if an error occurred).

SetActivityDueDate

This method will set a Due Datefor a specific Activity instance.

Input Parameters

ACTINSTID: The Timeline Activity Instance ID.

dateDue: The Date/Time value to represent the Activity's due Date.

Returns

Boolean: True if the operation succeeds.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean: True if the operation succeeds.

Start

This method will start an instantiated Timeline Instance.

Input Parameters

TLINSTID: The ID of the Timeline Instance to start.

Returns

TimelineInstance: The actual Timeline Instance for which the context was set.

wsUser

These web services enable you to manipulate Process Director Users pro-grammatically.

AddUserToGroup

This method will add a user to a specified group.

Input Parameters

UID: The UID of the user to add.

GID: The GID of the group to which to add the user.

Returns

Boolean: True if the operation succeeds.

AddUserToWorkspace

This method will add a user to a specified Workspace.

Input Parameters

UID: The UID of the user to add.

PROFILEID: The ProfileID of the Workspace to which to add the user.

Returns

Boolean: True if the operation succeeds.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

CreateExternalUser

This method will create an external user.

Input Parameters

UserID: The User ID to add.

Email: The email address of the user.

UserName: The name of the user.

GUID: The external GUID of this user record.

UserType: The authentication type for this user.

Returns

User: The new user object that was created, or null if an error occurred.

CreateExternalUser2

This method will create an external user.

Input Parameters

UserID: The User ID to add.

Email: The email address of the user.

UserName: The name of the user.

GUID: The external GUID of this user record.

UserType: The authentication type for this user.

Returns

User: The new user object that was created, or null if an error occurred.

CreateUser

This method will create a user.

Input Parameters

UserID: The User ID to add.

Email: The email address of the user.

UserName: The name of the user.

Password: The password of the user.

MustChangePassword: True if the user must change password on first login.

Returns

User: The new user object that was created, or null if an error occurred.

CreateUserInGroup

This method will create a user, and add the user to a specified Group.

Input Parameters

UserID: The User ID to add.

Email: The email address of the user.

UserName: The name of the user.

Password: The password of the user.

MustChangePassword: True if the user must change password on first login.

Group: The Group to which to add the user

Returns

User: The new user object that was created, or null if an error occurred.

DelegateUser

This method will delegate a user's tasks to another user.

Input Parameters

SrcUID: The UID of the source user.

DelegateToUID: The UID of the user to whom to delegate.

Returns

Boolean: True if the operation succeeds.

DeleteUser

This method will delete a user.

Input Parameters

UID: The UID of the user to delete.

Returns

Boolean: True if the operation succeeds.

DisableUserAccount

This method will disable the specified user's account and cancel the user in any active Workflow tasks.

Input Parameters

UID: The UID of the user whose account will be disabled.

ReplacementUID [optional]: The UID of a user who will be assigned in place of the disabled user.

Returns

Boolean: True if the operation succeeds.

EnableUserAccount

This method will enable the specified user's account.

Input Parameters

UID: The UID of the user whose account will be enabled.

Returns

Boolean: True if the operation succeeds.

GetUserByExtID

This method will get a User from its external ID field.

Input Parameters

ExtID: The external ID of the user.

Returns

User: The actual user or null if not found.

GetUserByID

This method will get a User from its UID.

Input Parameters

UID: The UID of the user.

Returns

User: The actual user or null if the user isn't found.

GetUserByUserID

This method will get a User from its UserID.

Input Parameters

UserID: The UserID of the user.

Returns

User: The actual user object or null if the user isn't found.

LockUserAccount

This method will lock the specified user's account.

Input Parameters

UID: The UID of the user whose account will be locked.

Returns

Boolean: True of the operation succeeds.

RemoveUserFromAllGroups

This method will remove a User from all user groups on the system.

Input Parameters

UID: The ID of the user to remove.

Returns

Boolean: True if the operation succeeds.

RemoveUserFromGroup

This method will remove a User from all user groups on the system.

Input Parameters

UID: The ID of the user to remove.

GID: The GID of the group from which to remove the user.

Returns

Boolean: True if the operation succeeds.

RemoveUserFromWorkspace

This method will remove a user from a specified Workspace.

Input Parameters

UID: The UID of the user to remove.

PROFILEID: The ProfileID of the Workspace from which to remove the user.

Returns

Boolean: True if the operation succeeds.

> ⛔ **This function will also return "false" if the user is only implicitly in the workspace, i.e., only in it because he is part of a Group that is assigned to the Workspace, rather than being assigned to it directly as a Workspace user. If you want to remove an implicit user from a workspace, you have to remove the user from the group, or remove the group from the workspace.**

ReplaceUser

This method will replace the user (UID) with another user (ReplacementUID) throughout the system.

Input Parameters

UID: The UID of the user whose account will be replaced.

ReplacementUID: The UID of a user who will be assigned in place of the disabled user.

Returns

Boolean: True if the operation succeeds.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean: True if the operation succeeds.

UnDelegateUser

This method will stop the delegation of a user's tasks.

Input Parameters

UID: The UID of the user for whom to stop delegation.

Returns

Boolean: True if the operation succeeds.

UnlockUserAccount

This method will unlock the specified user's account.

Input Parameters

UID: The UID of the user whose account will be unlocked.

Returns

Boolean: True if the operation succeeds.

UpdateUserFields

This method will update a user's information in the database. This method won't save any blank fields contained in the user's record.

Input Parameters

pUser: A User class containing the user's ID and the information to be updated.

Returns

Boolean: returns "true" if the operation succeeds.

UpdateUserLastActivityTime

This method will update the user object so set the current time as the last activity time for that user.

Input Parameters

UID: The UID of the user to update.

Returns

DateTime: The DateTime value of the time the user activity was updated.

UpdateUserRecord

This method will update a user's information in the database.

Input Parameters

pUser: A User class containing the user's ID and the information to be updated.

ignoreEmptyFields: A boolean that, when set to "false", will save all empty values in the pUser parameter into the database (normally these are ignored).

Returns

Boolean: Returns "true" if the operation succeeds.

wsUtil

These web services enable you to perform various utility functions pro-grammatically.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

ConvertSysVarsInString

This method will convert system variables in a given string.

Input Parameters

pString: The string to evaluate

Returns

String: The converted string.

GetUserTasks

This method will retrieve the task list for a user.

Input Parameters

PID: The Partition ID in which to locate the tasks. You may pass an empty string to retrieve from all partitions.

User: The UserID of the user whose task list to return.

Returns

String: An array of URL strings for task list entries.

GetUserTasksByEmail

This method will retrieve the task list for a user. This is most commonly used for unauthenticated users, who are only identifiable by email address.

Input Parameters

PID: The Partition ID in which to locate the tasks (pass the empty string for all partitions).

Email: The email address of the user whose task list to return.

Returns

String: An array of URL strings for task list entries.

GetUserTasksByUID

This method will retrieve the task list for a user.

Input Parameters

PID: The Partition ID in which to locate the tasks (pass the empty string for all partitions).

User: The UID of the user whose task list to return.

Returns

String: An array of URL strings for task list entries.

ImportGlobalKViewXML

This imports an XML package into the Global Knowledge View list.

Input Parameters

XMLData: The bytes of the XML document to import.

Returns

RetMsg: The list of output strings after the import.

ImportProfilesXML

This imports an XML package into the Profiles.

Input Parameters

XMLData: The bytes of the XML document to import.

Returns

RetMsg: The list of output strings after the import.

LoadBalancedRequest

This method is called when a load-balanced system performs some action.

Input Parameters

Action: The description of the action to perform

Return

Success: A Boolean value reflecting the success of the web service (FALSE if errors, TRUE if successful).

SequenceNumber

This method will get a unique sequence number.

Input Parameters

Group: The optional group from which the sequence is generated.

Returns

The unique sequence number or 0 if an error occurs.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls

Returns

Boolean: True if the operation succeeds.

Version

This method will return the server version.

Input Parameters

None.

Returns

Version: The server version number.

VersionString: The server version string.

wsWorkflow

> ⓞ **The Workflow object is the legacy process model used in early versions of Process Director. BP Logix recommends the use of the Process Timeline object, and not the Workflow object. The Workflow object remains in the product for backwards compatibility, but doesn't receive any new functionality updates, other than required bug fixes. No new features have been added to this object since Process Director v4.5. All new process-based functionality is solely added to the Process Timeline.**

These web services enable you to manipulate Process Director Workflows programmatically.

AddToWorkflow

This method will add an object to a Workflow instance.

Input Parameters

WFINSTID: The ID of the Workflow Instance.

ID: The ID of the object to add.

Type: ObjectType - The type of object to add.

Group: The optional group name to add the object into.

Returns

WorkflowInstance: The Workflow instance which the call ran (or null if an error occurred).

AddUsersToStep

This method will add user(s) to a specific step instance.

Input Parameters

STINSTID: The Workflow Step Instance ID.

UID: Comma separated list of UIDs to add.

Returns

Boolean: True if the operation succeeds.

Authenticate

This method will authenticate web service requests. Call this method prior to other web service calls. It will automatically populate the SOAP header with a validation token that enables subsequent web service calls.

Input Parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

Boolean: True if the operation succeeds.

AuthenticateJSON

This method will authenticate web service requests. Call this method prior to other web service calls. The return is a JSON string with the session ID.

Input parameters

User: The built-in or Windows UserID to use for the context of all calls.

Password: The password for this UserID.

Returns

JSON: The Session ID.

Cancel

This method will cancel a running Workflow instance.

Input Parameters

WFINSTID: The ID of the Workflow Instance

Returns

WorkflowInstance: The actual Workflow instance that was canceled, or Null if not found.

GetWorkflowByWFID

This method will get a Workflow definition from its WFID.

Input Parameters

WFID: The ID of the Workflow Definition

Returns

WorkflowDefinition: The actual Workflow definition, or null if not found.

GetWorkflowByWFINSTID

This method will get a Workflow instance from its WFINSTID.

Input Parameters

WFINSTID: The ID of the Workflow Instance.

Returns

WorkflowInstance: The actual Workflow instance, or null if not found.

GetWorkflowStepByName

This method will return a Workflow Step for a specific Workflow instance.

Input Parameters

WFINSTID: The Workflow Instance ID.

StepName: The Workflow Step name to retrieve.

Returns

WorkflowStep: The Workflow Step, or null if an error occurred.

GetWorkflowStepBySTID

This method will return a Workflow Step for a specific Workflow instance.

Input Parameters

WFINSTID: The Workflow Instance ID.

STID: The Workflow Step ID to retrieve.

Returns

WorkflowStep: The Workflow Step, or null if not found.

Instantiate

This method will instantiate a Workflow Instance from a WFID.

Input Parameters

WFID: The ID of the Workflow Definition to instantiate.

InitiatorUID: The optional UID of the Workflow initiator.

Returns

WorkflowInstance: The Workflow instance which the call instantiated, or null if an error occurred.

PostEvent

This method will post an event to a Workflow that is waiting on a Wait step.

Input Parameters

WFINSTID: The optional ID of the Workflow Instance to POST.

EventName: The optional Event Name to post.

Returns

Boolean: True if the operation succeeds.

RemoveUsersFromStep

This method will remove user(s) from a specific step instance.

Input Parameters

STINSTID: The Workflow Step Instance ID.

UID: Comma separated list of UIDs to add.

Returns

Boolean: True if the operation succeeds.

Restart

This method will restart an existing Workflow instance WFINSTID at an optional step STID.

Input Parameters

WFINSTID: The ID of the Workflow Instance to restart.

STID: The STEP ID of the STEP in the Workflow definition to restart (optional).

Returns

WorkflowInstance: The Workflow instance which the call restarted, or null if an error prevents execution of the operation.

Run

This method will instantiate and run a Workflow Instance from a WFID.

Input Parameters

WFID: The ID of the Workflow Definition to run.

Returns

WorkflowInstance: The Workflow instance which the call ran , or null if an error prevents execution of the operation.

RunWorkflow

This method will create and run a new Workflow instance and optionally add a Workflow object.

Input Parameters

PID: The Partition ID or Name.

PathName: The full path of the Workflow definition (e.g. /Workflows/My Workflow).

ID: The ID of the object to add.

Type: ObjectType - Place "NotSet" in this field.

Group: The optional group name to add the object into.

InitiatorUID: The optional UID of the Workflow initiator.

Returns

WorkflowInstance: The Workflow instance which was created , or null if an error prevents execution of the operation.

RunWorkflowWithForm

This method will create a new form instance, a new Workflow instance and attach the form.

Input Parameters

PID: The Partition ID or Name.

WorkflowPathName: The full path of the Workflow definition (e.g. /Workflows/My Workflow).

FormPathName: The full path of the form definition (e.g. /Forms/My Form).

FormValues: The list of name/value pairs to set in the new form instance.

SkipDefaultValues: Set to true to skip merging default form values the first time the form is viewed.

Group: The optional group name to add the object into.

InitiatorUID: The optional UID of the Workflow initiator.

Returns

WorkflowInstance: The Workflow instance which was created , or null if an error prevents execution of the operation.

SetContextUID

This method will set the context UID for all future web service calls. For instance, this UID will be used as the current user for running Knowledge Views, permission checking, etc.

Input Parameters

UID: The ID of the user to use for the context of all calls.

Returns

Boolean: True if the operation succeeds.

SetStepDueDate

This method will set a Due Date for a specific step instance.

Input Parameters

STINSTID: The Workflow Step Instance ID.

dateDue: The Date/Time value to represent the Step's due Date.

Returns

Boolean: True if the operation succeeds.

Start

This method will start an instantiated Workflow Instance.

Input Parameters

WFINSTID: The ID of the Workflow Instance to start.

Returns

WorkflowInstance: The actual Workflow Instance.

REST Services

REST (Representational State Transfer) is a standards-based data architecture for sharing data between computers, using web-based access. Services that use REST are often called RESTful services. A REST service usually transfers data from a server-based data store to a client machine that requests it in a specified format—usually via a specifically configured URL.

The important thing to remember about REST services is that the server and client machines are unrelated. The activities on the client are completely unaffected by the activities of the server, and vice versa. Unlike a connection to a database like SQL server, the client machine does not need to store a connection string or maintain a constant connection to the server. Similarly, any changes to the code or operations of either machine have no effect on the other.

The only connection that occurs between the two systems is when the client machine sends a request to the server via URL, and the server responds by sending back data in a REST-compliant format. This "connection" is also **stateless**, which is to say that the server and client machines know nothing about each other, outside of a specific REST transaction. A **REST transaction** consists of a **request** from the client, and a **response** from the server that contains the requested data. The REST request must provide all of the information necessary for the server to respond to the request properly. The server cannot use any knowledge of past requests to complete the current request and return a response. Each transaction is made as if it was an entirely new request, unrelated to any previous requests.

REST is not a programming format. It is purely a set of architectural guidelines for the communications format used between two systems. As long as each machine knows which format to use for requests and responses, data can be shared. Because REST is an architectural construct, it is language-independent. REST services can be constructed in any programming language, and the programming language used by the client and server are irrelevant to each other. As long as both systems use the correct architecture for their communication, they can share data.

The REST Request

A **REST Request** is a message sent to a server that asks for a response containing data. In most cases, a REST request is sent to an available URL. The URL may be publicly available as a service on the internet, or it may be available only inside the same network as the client. As long as the client can access the URL, however, it can request a REST response from the server.

The URL contained in the REST request is usually formatted in a specific way, in order to match the way in which the REST service is implemented on the server. For example, the REST service may require the URL be formatted hierarchically. A notional request URL for a publicly available REST Service that returns the current weather might be:

`https://myrestservice.com/current/zip/92111`

The server would interpret this request as asking for the current weather for the 92111 Zip Code.

Alternatively, the URL for the same information might require a URL that uses parameters, such as:

`https://myrestservice.com?type=current&zip=92111`

Both of these formats can constitute valid requests, as long as the server understands the format.

The REST Request may be more complex than shown above, and require additional information such as login credentials, specific HTTP header information sent with the request, and Key/Value pairs to provide parameters, or include other information. In Process Director, much of this additional complexity is hidden, but a fully formatted REST request with header information and Key/Value pairs to provide parameters for the data to return might look like this.

```
POST https://myrestservice.com/weather HTTP/1.1
Host: myrestservice.com
Content-Type: application/json
Content-Length: 42
{"Type":"current","Zip":92111}
```

Business Values have a feature for adding authentication credentials and Key/Value pairs to the HTTP header, as described in the [Business Values topic](#) of the Implementer's Guide.

The REST Response [#](#)

Once the server receives a valid request, the **REST Response** returns the requested data. Even if there's no data to return, the server will still send a response, to note that the request was received and fulfilled. The REST response will be presented in one of two formats, JSON or XML.

Both XML and JSON formats are used to represent data, though each one is very different.

XML is a highly-structured data language. Each element of data is presented in nodes that are set off by the <> characters, e.g.:

```
<temperature    value="78.66"    min="70.56"    max="87.89"    unit-
t="fahrenheit"/>
```

JSON, on the other hand, is less highly structured than XML, and data is returned in a JavaScript-based format, e.g.:

```
"temperature": {"curr":78.66,"min":70.56,"max":87.89}
```

This format is not JavaScript, though it is based on the JavaScript language structure. Both JSON and XML are language-independent formats, and both are essentially formatted text files, though the formatting is, as we shall see later in this module, very important.

There are advantages and disadvantages between the two language formats.

- XML, because it's so highly structured, is very good at returning hierarchical data in a more easily comprehensible format. It's easier for humans to parse when, as we'll see later, we need to find specific data in an XML response. On the other hand, XML is relatively verbose, which means that, given the same data, an XML response might be significantly larger and take up more memory, than a JSON response.

- JSON, being less structured, is more flexible. A JSON response could structure the data in many different ways, while the highly structured nature of XML limits the way in which data can be returned. On the other hand, this flexibility means that two similar requests might return data that is structured slightly differently, meaning that the method you use to parse one request might not work on a similar one. Also, JSON's variable structure, and lack of a specific node structure can make JSON less readable and harder to parse by humans.

In general, JSON is a more popular format, due in part to its flexibility and versatility, but mainly because JSON responses are generally more lightweight than XML responses. There are still many REST services that return XML, or return both formats. In Process Director, both XML and JSON response formats can be used.

More Information about Rest Services

JSON and XML for REST: How sample XML or JSON REST responses might compare.

JSONPath and XPath: A comparison of the XPath and JSON Path syntax needed to parse XML or JSON REST responses.

JSON and XML for REST

To illustrate the different between JSON and XML data that might be returned from a REST service, let's look at a small data sample, and see how it would be represented in each language. For this example, we'll use a list of file names that might be returned by a REST Service:

- Image1.jpg
- Description.docx
- Forecast.xlsx
- Agreement.pdf
- NetworkDiagram.vsdx

XML

```xml
<?xml version="1.0"?>
    <Files>
        <item>Image1.jpg</item>
        <item>Description.docx</item>
        <item>Forecast.xlsx</item>
        <item>Agreement.pdf</item>
        <item>NetworkDiagram.vsdx</item>
    </Files>
```

The structured nature of XML limits how we can represent this data. We must have a "Files" node that contains all of the files. Each file needs its own "Item" node. Because XML is so highly structured, there are strict limits on how data can be represented.

JSON

JSON is much less structured than XML, and can present this same data in many ways.

Single Elements

```json
{
    "Image":"Image1.jpg",
    "Document1":"Description.do-
cx",
    "Spreadsheet":"Forecast.xls-
x",
    "PDF":"Agreement.pdf",
    "Visio":"NetworkDiagram.vsd-
x"
}
```

Structured Array

```json
[
    {"Files":"Image1.jpg"},
    {"Files":"De-
scription.docx"},
    {"Files":"Fore-
cast.xlsx"},
    {"Files":"A-
greement.pdf"},
    {"Files":"Net-
workDiagram.vsdx"}
]
```

Unstructured Array Elements

```json
{
    "Files":
    [
        "Image1.jpg",
        "Description.docx",
        "Forecast.xlsx",
        "Agreement.pdf",
        "NetworkDiagram.vsdx"
    ]
}
```

Structured Array Elements

```json
{
    "Files":
    [
        {"f":"Image1.jpg"},
        {"f":"De-
scription.docx"},
        {"f":"Fore-
cast.xlsx"},
        {"f":"A-
greement.pdf"},
```

```
        {"f":"Net-
workDiagram.vsdx"}
    ]
}
```

While this flexibility is nice, it does mean that you need to know, to a much greater degree than with XML, how the specific data you get in a response is structured, in order to parse and use it properly.

From an implementer's point of view, you may find XML REST Services easier to work with, if available, especially if you're just starting to use REST services. The structured nature of XML generally makes it more readable, and easier to understand.

More Information about Rest Services

REST Services: An overview of using REST Web Services.

JSONPath and XPath: A comparison of the XPath and JSON Path syntax needed to parse XML or JSON REST responses.

JSONPath and XPath

In most cases, you'll use a Business Value to request and parse responses from REST Web services. In Process Director, the Business Value provides the easiest and most effective method for making REST requests and using the data that is included in the REST response.

Just like a Business Value that returns SQL data, a REST Business value will receive a response containing data field values. You must configure a Business Value property for each data field you wish to access from the REST Response.

For each property, the Property Name column is where you'll provide a name for each property, which will be used in the Process Director UI. For each field in the REST Response, the Property Type must be set to *REST Data* in order to extract the data from a single data field in the REST Response.

An additional setting, *Entire REST Response* will, as the name implies, simply use the entire XML or JSON file content as the property value. This setting may be useful if you need to see the REST Response contents to understand how to parse them to extract individual fields.

Parsing the REST Response is very important to ensure you extract the correct data. The Property Value column is where you provide Process Director with the instructions on which REST Response field you wish to extract for each property.

Unlike a SQL Business Value, where you simply write the name of a valid database field in this column, a Rest Business Value requires that you provide detailed instructions for parsing the REST Response, to extract the data you want.

The parsing syntax you'll need to use will vary, based on whether the REST Response contains data in XML or JSON Format. Parsing an XML REST response requires you to use a parsing format called **XPath**, while parsing a JSON response requires the use of **JSONPath**.

Time and space do not allow us to provide detailed training on these two parsing formats here. There are many freely available resources on the Internet for learning the basics of both XPath and JSONPath. Here are two resources you might find useful, but there are many others:

- XPath: W3Schools.Com
- JSONPath: SmartBear.com

> ⓘ **BP Logix does not endorse these resources. They are merely shown here for your convenience.**

Let's look at some simple XML and JSON REST Responses to see how we'd parse them using each format. For this example, we'll use some sample weather data for San Diego, CA. These REST documents are available on the BP Logix Documentation portal. Those these are static documents, you can use their URL as the REST URL of a Business Value, for testing on your Process Director installation.

- XML REST Response
- JSON REST Response

You can use the URL for either of these documents

XML and XPath

The XML REST Response looks like this:

```
<current>
    <city id="0" name="San Diego">
        <coord lon="-117.2028" lat="32.7635" />
```

```
        <country>US</country>
        <timezone>-25200</timezone>
        <sun rise="2023-04-07T13:29:04" set="2023-04-
08T02:12:27" />
    </city>
    <temperature value="66.67" min="59.76" max="74.79" unit-
t="fahrenheit" />
    <feels_like value="65.57" unit="fahrenheit" />
    <humidity value="54" unit="%" />
    <pressure value="1017" unit="hPa" />
    <wind>
        <speed value="12.66" unit="mph" name="Moderate breeze"
/>
        <gusts />
        <direction value="270" code="W" name="West" />
    </wind>
    <clouds value="75" name="broken clouds" />
    <visibility value="10000" />
    <precipitation mode="no" />
    <weather number="803" value="broken clouds" icon="04d" />
    <lastupdate value="2023-04-07T21:39:25" />
</current>
```

XPath uses the forward slash character (/) to navigate from the highest node to the lowest one. The `current` node is the top node of the XML REST response, and it contains the `city`, `clouds`, and `temperature` nodes.

Part of XML's structured nature is to name each node in the XML structure. At the very top of the response, the first node is named `<current>`. Since we have to start our XPath expression at the highest node, and we need to add a slash between each node, our XPath expression will begin with:

`current/`

If we want to find the XML node that tells us whether it's clear or cloudy, we'd need to add the `<clouds>` node, which is located in the second level of the XML hierarchy:

`current/clouds/`

The `<clouds>` node contains two attributes: `value` and `name`. We need to get the data in the `name` attribute. Attributes are identified using the "At sign" (@), which means that this value is an attribute of the previous node. So, we need to add this value to our expression as:

`current/clouds/@name`

This gives us the full XPath expression to access our data, which should return the value `broken clouds`.

We can use different XPath expressions to return other data from this response, as shown in the table below.

Data	XPath Expression
City	`current/city/@name`
Country	`current/city/country`
Current Temp	`current/temperature/@value`
Low Temp	`current/temperature/@min`
High Temp	`current/temperature/@max`

JSON and JSONPath

The JSON REST Response looks like this (Though the formatting makes it look like the JSON Response contains more information, it is the same length, 859 characters, as the XML response shown above):

```
{
    "coord": {
        "lon": -117.2028,
        "lat": 32.7635
    },
    "weather": [
        {
            "id": 803,
            "main": "Clouds",
            "description": "broken clouds",
            "icon": "04d"
        }
    ],
    "base": "stations",
    "main": {
        "temp": 66.65,
        "feels_like": 65.55,
        "temp_min": 59.76,
        "temp_max": 74.79,
        "pressure": 1017,
        "humidity": 54
    },
    "visibility": 10000,
    "wind": {
        "speed": 12.66,
        "deg": 270
```

```
    },
    "clouds": {
        "all": 75
    },
    "dt": 1680903688,
    "sys": {
        "type": 2,
        "id": 2017332,
        "country": "US",
        "sunrise": 1680874144,
        "sunset": 1680919947
    },
    "timezone": -25200,
    "id": 0,
    "name": "San Diego",
    "cod": 200
}
```

JSON path uses a dollar sign character (`$`) to denote the top or global level of the JSON response. Each level below that is separated by a period (`.`)character.

In JSON, each data node in the hierarchy is marked by the use of either curly brackets (`{}`)or square brackets (`[]`). In general, when counting from the top of the Response, each time we reach a bracket character, we need to use a period (`.`) to specify a different node.

The global JSON Response is always specified by using the dollar sign (`$`) character. So, that's always the first character of a JSONPath parsing phrase. At the very top of the response is the opening curly bracket that contains the full response. This curly bracket starts the first node of the response, and we have to add a period to the JSONPath expression to indicate we are moving to the next lower node in the hierarchy:

`$.`

At this level of the response, there are two nodes, `coord` and `weather`. If we once again want to extract the data that indicates whether it's clear or cloudy, the data we need is in the `weather` node, so we need to add it to the JSONPath expresson:

`$.weather`

The weather portion of the JSON response contains several lower-level nodes. The actual data for current conditions is contained in the `main` node of the `weather` portion. But note that, immediately after `weather`, there is a square bracket, followed by a curly bracket in the JSON response. Each of these brackets requires that

we add a period character as a node marker for each of them, in order to properly count the nodes between `weather` and `main`. Thus, we end up with our completely parsed path of:

`$.weather..main`

This gives us the full JSONPath expression to access our data, which should return the value `Clouds`.

We can use different JSON Path expressions to return other data from this response, as shown in the table below.

Data	JSONPath Expression
City	`$.name`
Country	`$.sys.country`
Current Temp	`$.main.temp`
Low Temp	`$.main.temp_min`
High Temp	`$.main.temp_max`

More Information about Rest Services

REST Services: An overview of using REST Web Services.

JSON and XML for REST: How sample XML or JSON REST responses might compare.

This page intentionally left blank.

Index

.

H

O

P

R

S

V

W

This page intentionally left blank.