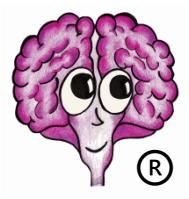
PTI-TD-00016
Revision 11.1

DocuBrain® TechDoc Workflow Guide



A DocuBrain® Product

https://docubrain.com/

By Prevo Technologies, Inc.

https://prevo.com/

DocuBrain® TechDoc Workflow Guide

By Prevo Technologies, Inc.

Published by Prevo Technologies, Inc., 1111 Keener Rd, Seymour, TN, 37865.

Table of Contents

Revision History

Revision	Date	Comments
11.1	02/07/2025	Initial TechDoc 11.1 Workflow Guide Release

1. Introduction

Workflow is a term that is used to describe all of the steps, tasks, or events that occur in a business process and the participants or organizations involved. For example, think of an automobile manufacturer (the organization) and all of the steps involved in the manufacturing process of a car: receiving raw materials, casting and machining parts, assembly of the engine and transmission, construction of the chassis, etc. Though complex, every process can be broken down to individual steps.

In order to notate a process, each step must be identified along with the requirements and participants involved. Once a process has been identified, it can be modeled. The most widely used and accepted method of describing a workflow process is Business Process Model and Notation (commonly referred to as BPMN). Using BPMN (usually through a graphical design tool such as the DocuBrain® Workflow Editor), an individual can map out each step of a process from start to end. Once the process has been modeled using BPMN, it can be uploaded to a workflow engine and executed.

TechDoc features a vast set of capabilities, with one of the main features being a workflow engine. The workflow engine included in TechDoc is BPMN 2.0 compliant and has been tightly integrated with TechDoc to provide a seamless workflow experience. While you can design processes to be completely standalone and independent from TechDoc, you can also perform a large array of TechDoc operations through BPMN Service Tasks as described in the TechDoc Activities section below.

Before you dive in and start trying to build and design workflow processes, it is necessary to become familiar with the BPMN 2.0 specification. Throughout this guide, we will touch on small pieces of the specification and how they pertain to workflow in TechDoc however, we will not be going into depth reiterating the BPMN 2.0 specification.

2. Getting Started

This guide contains the information you need to understand to design, build, and manage workflow processes within DocuBrain products. Starting from the ground up, we will cover from the fundamentals of process design to deployment and execution on the TechDoc workflow engine.

- Overview of Features – Provides a high-level overview of the features and capabilities.
- DocuBrain Workflow Editor Tutorial – A tutorial detailing the use of the DocuBrain Workflow Editor.
- TechDoc Service Task Operations – Covers the TechDoc specific operations that are available for use via Service Tasks in a workflow process.
- The TechDoc Workflow Engine – Overview of the TechDoc workflow engine and how to interact with it.

3. Overview of Features

DocuBrain TechDoc features a complete workflow management system. From process design to execution and management, we have you covered. In today's every changing business world, business process automation is invaluable. Using DocuBrain TechDoc, you can leverage all the benefits of electronic document and records management while streamlining all of your business processes. Integration of electronic document management and business process automation has never been easier.

3.1. *DocuBrain Workflow Editor*

The DocuBrain Workflow Editor is an easy to use and intuitive workflow process editor. It allows users the ability to graphically model business processes following the BPMN standard and deploy them directly to a TechDoc workflow engine. The DocuBrain Workflow Editor can be used in conjunction with TechDoc or used on its own to create generic BPMN 2.0 workflow processes. The DocuBrain Workflow Editor features:

- Drag and drop modeling
- BPMN 2.0 compliant
- Code free process design
- Customizable look & feel
- Connects to TechDoc Document Managers
- Multi-diagram deployments
- Collaborative process modeling
- Process repository
- Process versioning
- BPMN fragments/snippets
- Form editing
- Process deployment
- Model validation

- Diagram printing

3.2. *Workflow Engine*

The TechDoc workflow engine is an extremely fast and lightweight Business Process Management (BPM) Platform. The process engine is BPMN 2.0 compliant and fully integrated with TechDoc. Alongside all of the abilities of BPMN 2.0, an extensive set of TechDoc operations are also available for use via BPMN Service Tasks. In addition, workflow processes can be triggered to execute based on different events in a TechDoc Document Manager such as the creation, modification, or deletion of a document. Here a just a few features of the TechDoc workflow engine:

- Completely web-based, accessible from anywhere
- Process administration and management dashboard
- Real time monitoring with graphical view of workflow process and process variable inspector
- User task integration with TechDoc Users and Groups
- Email-based notification and alerts
- Direct launch of user task completion from email notification
- Process versioning
- Centralized exception handling
- Inter-process communications
- Parallel activities (forking and joining)
- Long running workflows (months or more)
- Sub-process and task looping
- Expression evaluation using Unified Expression Language (UEL)
- Escalation
- Forms support
- History logging

- Reporting

3.3. *Electronic Notifications*

The workflow engine is fully integrated with TechDoc's email and notification system. The engine sends email notification to users when: a process requires them to complete a task, a process of theirs encounters a business exception or escalates, etc. Additionally, the email notification sent contains action links to the TechDoc Document Manager that will take the user straight to the item needing their attention. This means you can stay connected even while on the go with your smart phone or tablet.

3.4. *User Task Management*

User Tasks generated by workflow processes are completely integrated with TechDoc. User Tasks appear under the TechDoc Document Manager My Work area just like all other TechDoc items that require a user's attention. Additionally, as mentioned in the previous section, User Tasks also notify the user via email when a task needs to be completed.

What happens when a User Task is encountered within a process, and the user assigned is gone on vacation? No worries here, you can easily locate and re-assign a User Task under the Workflows Management section in the TechDoc Document Manager. Furthermore, User Tasks can also be un-assigned so they are made available to any other potential owners, or can just be completed directly without the need to be re-assigned. Flexibility is everything with User Task management.

3.5. *Monitoring and Administration*

Monitoring the TechDoc workflow engine could not be easier. Under the Workflows Management section of a TechDoc Document Manager, a user with the Workflows Management privilege can easily view all of the deployments, process definitions, and process instances in the system as well as all User Tasks awaiting completion.

Have you ever wondered exactly what step your process instance is currently on? Many systems simply tell you the name or ID of the element being executed and then require you to pull up the diagram and try to locate the element yourself. In TechDoc, you can simply click a running process instance. TechDoc features a real-time interactive process diagram so you can see the process just as it was designed as well as the activity being executed. However, we did not stop there. You can also click elements on the process and view all of their metadata in a popup window. Have you ever forgotten how long of a period you set on a timer task or the assignee of a User Task? Just click the activity and

take a quick peek; no more firing up an editor and loading a diagram just to figure it out. Additionally, when a process instance is focused on a User Task, you can just click the User Task and you will be taken straight to that task to complete.

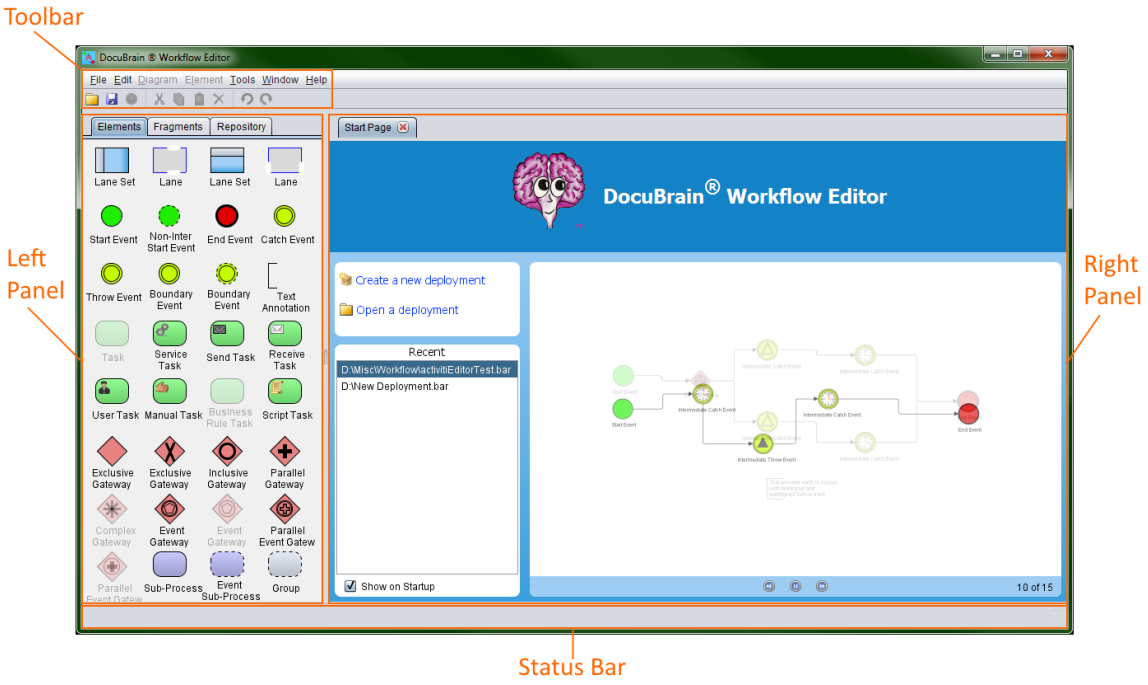
All of this is great for administrators, but what about normal users? All users can make use of the real-time monitoring and interaction with the only restriction being they can only view processes and assign/complete tasks they own.

4. DocuBrain Workflow Editor Tutorial

In this section, we will go over all the basics of using the DocuBrain Workflow Editor. We will not touch much on the BPMN specification, but there will be portions that go over interaction with the BPMN elements. As the BPMN elements do differ a great deal, it is good idea to at least be familiar with the basic elements: Start/End Events, Intermediate Events, Boundary Events, Tasks, and Gateways.

4.1. *Layout of the User Interface*

The editor is split into four main areas: the Toolbar, Left Panel, Right Panel, and Status Bar.



4.1.1. Toolbar

On the toolbar, you will find all of the menus for the editor and the short cut buttons. The toolbar contains the following menus:

- File – Used to open, close, and save workflow deployments as well as add, remove, import or export diagrams from an open deployment. Additionally, this menu can be used to print an open deployment or upload it to a workflow repository.

- Edit – This menu is only used when a deployment is open. It contains the undo/redo options to undo or redo an action to a diagram such as un-delete an element. This menu also contains all of the clipboard operations: to cut, copy, paste and delete elements from or to a diagram.

- Diagram – This menu is only used when a deployment is open. It contains all of the properties for the currently selected diagram. This menu is also accessible by simply right clicking on the diagram (grid paper). On this menu, you will find options like setting the file name of the diagram, the BPMN process ID and name, editing the XML namespaces for the diagram, etc.

- Element – This menu is only used when a deployment is open and a BPMN element is selected on a diagram. It contains all of the properties for the currently selected BPMN element. This menu is also accessible by simply right clicking on an element on your diagram. On this menu you will find the properties for the element, the ability to cut or copy the element (or selection if more than one element is selected), and other options that vary from element to element depending on its type.

- Tools – The Tools menu contains all of the tools and global settings for the editor. This is where you will find the Options dialog where you can change the target workflow engine and edit presets. You can also stylize the Diagram Editor and XML Editor to suit your needs. Additionally, the XML Editor's theme can be changed if needed. Note that there is a high contrast theme for those who need it.

- Window – The Window menu contains all of the options to display the various windows of the editor if they have been closed such as the Start Page.

- Help – The Help menu contains the About dialog that shows the editor's version and your license info, as well as the options to check for updates to the editor and submit feedback on the editor to the DocuBrain website.

The icons of the toolbar from left to right are shortcuts to toolbar menu items. There are as follows:

-  File-> Open Deployment

-  File-> Save Deployment

- File -> Upload Deployment
- Edit -> Cut
- Edit -> Copy
- Edit -> Paste
- Edit -> Delete
- Edit -> Undo
- Edit -> Redo

4.1.2. Left Panel

The left panel contains four tabs: Deployment, Elements, Fragments, and Repository tab.

The Deployment tab displays the deployment that is currently open in the editor. Each process contained in the deployment is listed here and may be opened by double clicking it. Additionally, you can right click a process and delete it from the deployment or right click the deployment and add a new process.

On the Elements tab, you will find all of the BPMN elements that the editor currently supports. To drag an element onto a diagram, press and hold the left mouse button over an element, drag it over to the diagram and release the left mouse button. This will place the element onto the diagram; you can then proceed to make connections, size and position the element, etc.

On the Fragments tab, you will find all of the fragments (or snippets as they are sometimes referred to) that you have created. Fragments are not usually a full diagram, but just a few elements connected in a particular way that you find yourself re-creating repeatedly. We will dig deeper into fragments later on.

The Repository tab is used to connect to a TechDoc workflow engine. Once a connection has been configured, you can select it from the dropdown box and click the connect button to connect to the workflow engine and retrieve the list of all the workflow deployments and processes it contains. Once the list of deployments has been retrieved, you can select a deployment and click the download button to download the deployment from the workflow engine (repository) and load it in the editor to view or edit. A deployment can be deleted from the repository by selecting it from the list and clicking the delete button. When a deployment is selected from the list, you can

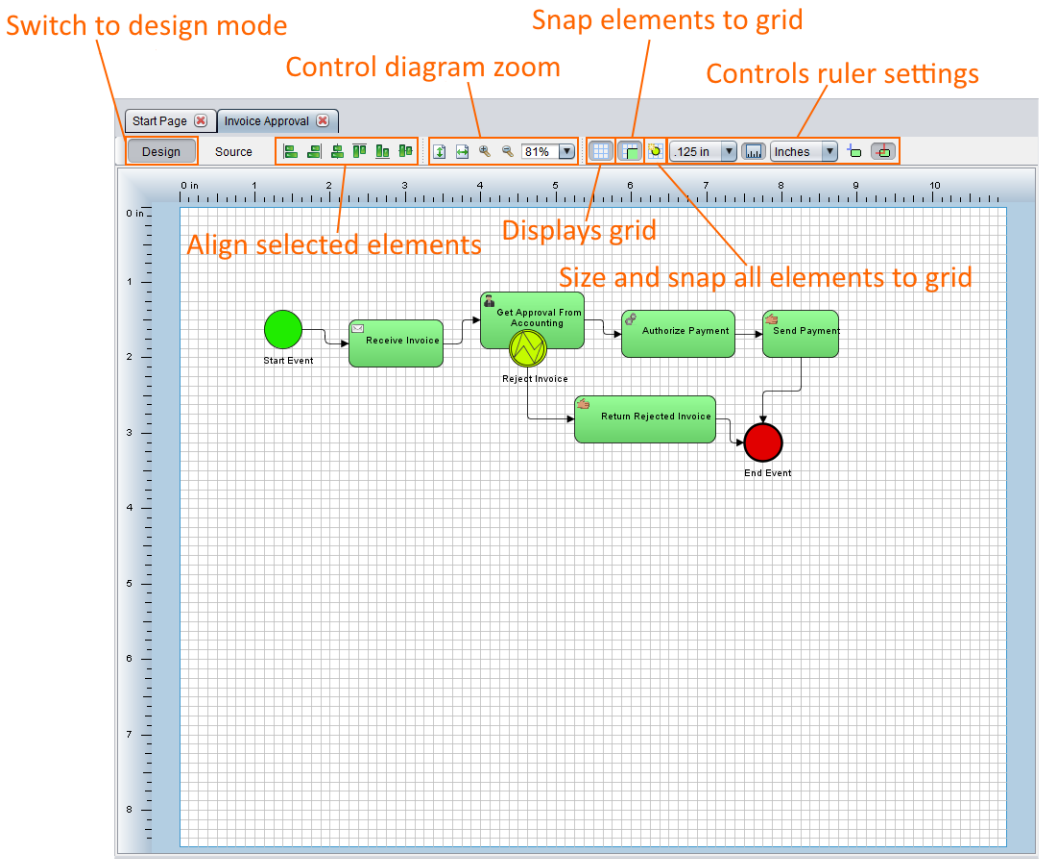
view the Definitions, Resources, and Images it contains using the tabs below the deployment list. We will go into more detail later on about repository connections.

4.1.3. Right Panel

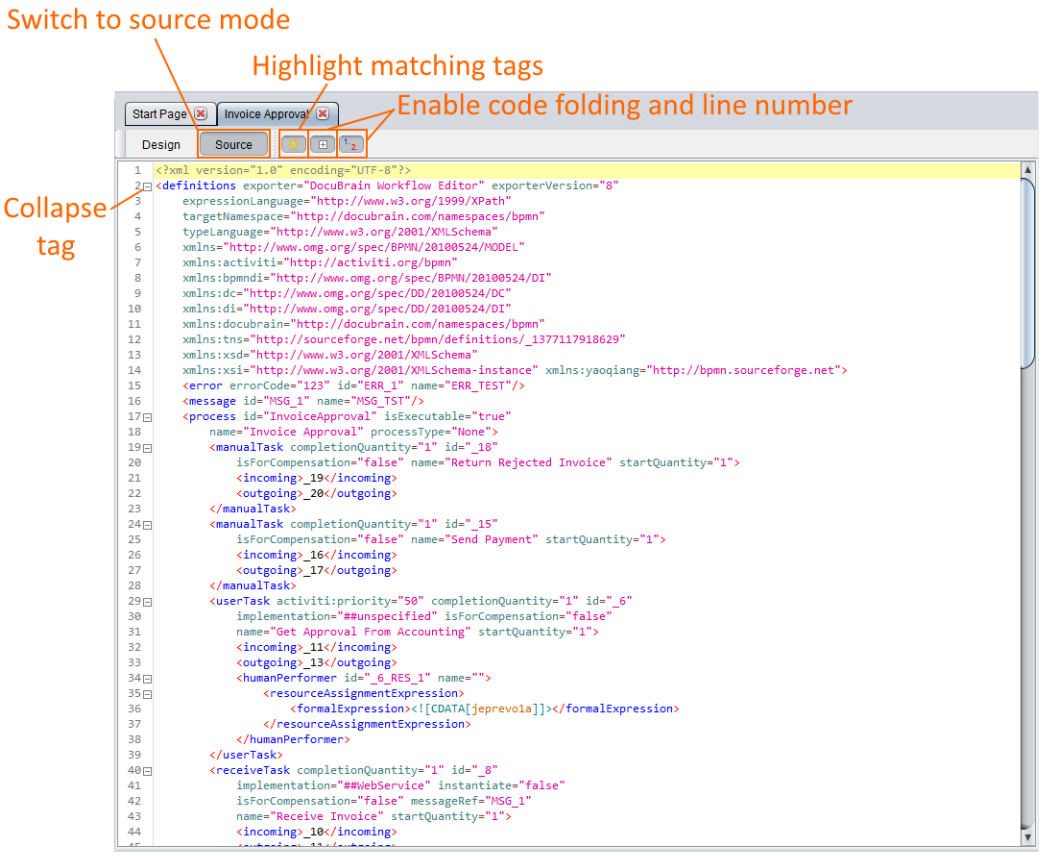
The right panel is the workspace of the editor. When you create or load a deployment, all of the diagrams are loaded into the workspace. Each diagram will have a tab, and on each tab, there are two main sections the Design view and Source view.

The design view is used to graphically layout a workflow process. Elements can be dragged from the elements panel on the left panel to the diagram. Once an element has been placed on the diagram, it becomes part of the workflow process. To view the properties of the diagram, right click anywhere on the graph paper. To view the properties of an element, right click on an element.

The design view features a both a ruler and tool bar. The ruler can be set to use inches, metric, or pixel-based measurements. The graph paper's grid size synchronizes to the ruler's settings and is very helpful when placing and aligning elements. The toolbar contains various buttons and settings, see the image below for a description of each:



The source view is used to examine the XML BPMN code generated by building the diagram in design mode. The source view is used purely for reference and cannot be used to alter or manually code BPMN. This view also contains a toolbar; see the image below for a description of each control:



4.1.4. Status Bar

The status bar is the bottom bar of the editor. It is used to display status messages when the editor is busy doing things such as saving or loading a deployment, retrieving a list of deployments from a repository, etc. The status bar can be very useful to watch the progress of action that can take just a bit to complete. For example, if you are connecting to a workflow repository over a slower internet connection that contains many deployments, it could take several seconds to a minute.

4.2. *Creating a Workflow Process*

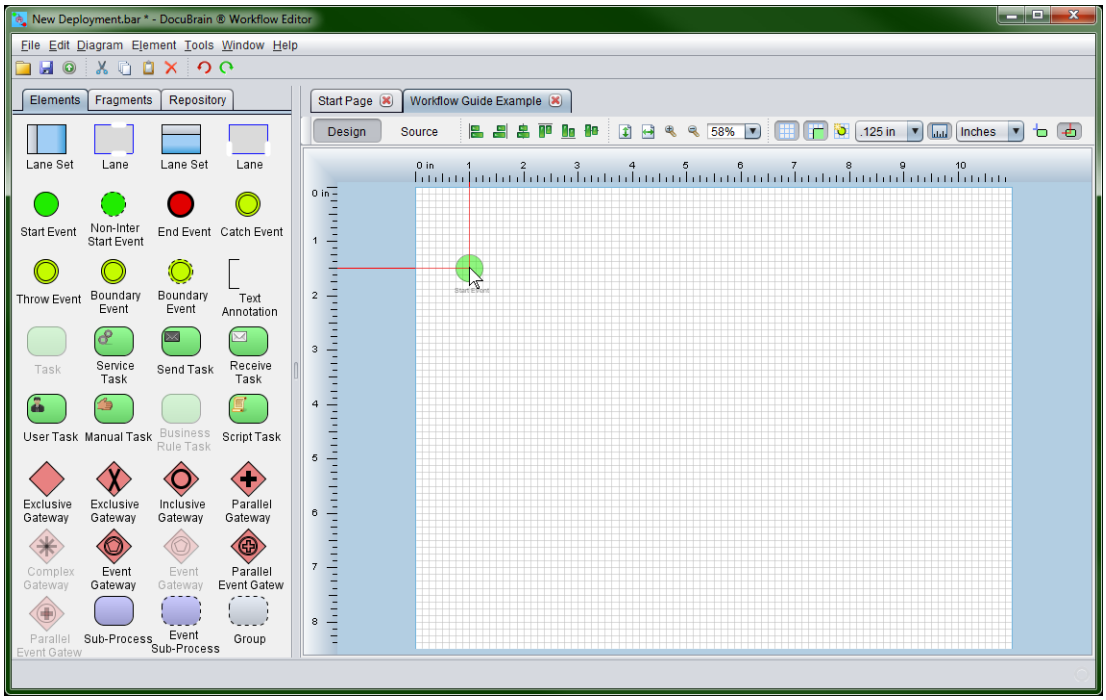
In this section, we will cover what you need to know to construct a basic workflow process. Start by launching the editor and selecting "New Deployment" from the File menu, or by clicking "Create a new deployment" on the Start Page. You will then be presented with a new deployment containing just one diagram.

When creating a new process, you must always start by giving the diagram a file name, and then name the process. Right click on the diagram (the graph paper), and then click

Diagram Properties under Diagram. On the Diagram Properties dialog, enter the filename to use for the diagram and then click OK.

Now right click on the diagram, and then click Process Properties under Diagram. On the Process Properties dialog, you will want to give the process its name and ID; this is a very important step. Later on, when you deploy this process to a workflow engine, it will be very hard to identify your process if it does not have a unique name. Additionally, if you upload a process with a name or ID that is already being used by another process, many workflow engines will assume this is a new version of the original process and may begin using it for unintended purposes. After you have given the process a name and ID, click the OK button.

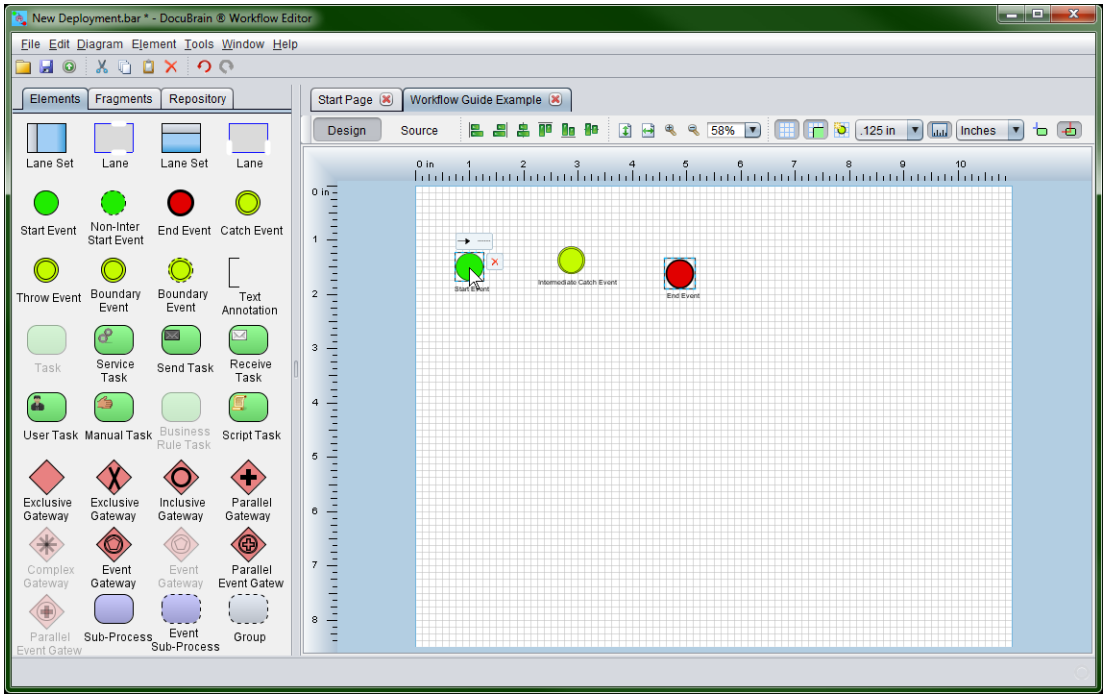
Now we have a named diagram file and process. Let us begin by dragging a Start Event from the elements panel on the Left Panel onto the diagram.



Notice when we drag the element around on the diagram, there is a red alignment line extending from the center of the element towards the ruler. These alignment lines can be used to make sure we place the elements exactly where we want it. If you prefer, you can click the corner align button (blue alignment line) from the upper toolbar so that you may align the element using its upper left corner.

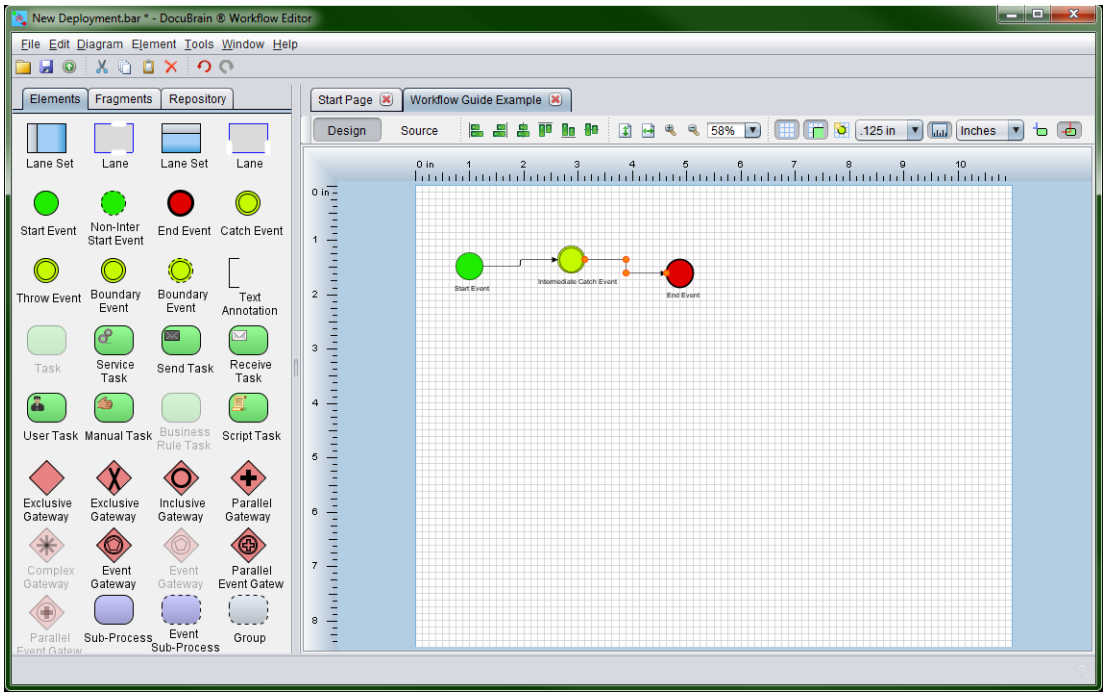
Now that we have placed a Start Event, go ahead and drag a Catch Event and End Event onto the diagram as well. In order for a workflow engine to know how to execute your process, the elements need to be connected in the order in which they should execute

beginning with the Start Event. Let us begin by connecting the Start Event to the Catch Event. To do this, click on the center Start Event to show its actions:



Once the actions appear, click and hold the left mouse button over the connection arrow. Now drag the red connection line over the center of the Catch Event and release the left mouse button to make the connection. This will make a Sequence Flow connection from the Start Event to the Catch Event. The order in which you make connections is important. Notice how the arrow on the line points to the Catch Event. This means the connection is sourcing from the Start Event and targeting the Catch Event. If we had done this backwards, the connection would be starting from the Catch Event and ending on the Start Event, which is not the direction that we intend for the engine to execute our process; in fact, this is also illegal, as a Start Event cannot have an incoming connection by the BPMN 2.0 specification.

Now that you have connected the Start Event and Catch Event, go ahead and connect the Catch Event to the End Event. You diagram should now look like this:



Notice the orange dots along the connection. If desired, you could left click and hold on top of an orange dot, drag it to a new location and release the mouse button to change the connection's path. Additionally, if you drag the first or last dot you can change the point at which the connection leaves or enters the source or target element.

Now we have a complete BPMN process. By the BPMN specification, to have a complete process, you must have a Start Event and an End Event. While this process would execute on a workflow engine, it would be near impossible to catch while it is running because as soon as you start it, it would complete because there are no operations to perform. Let us address this by giving the Catch Event a timer definition. Right click on the Catch Event and then click Event Definitions.

On the Element Event Definitions dialog, click the create button and then select Timer under element defined. Select Duration as the type and enter PT1M as the expression to say that we want a duration period of 1 minute. Click the OK button on the Create Timer Event Definition dialog. Now on the Element Event Definitions dialog, you can see the timer event definitions that you just created. Click the OK button to close the Element Event Definitions dialog.

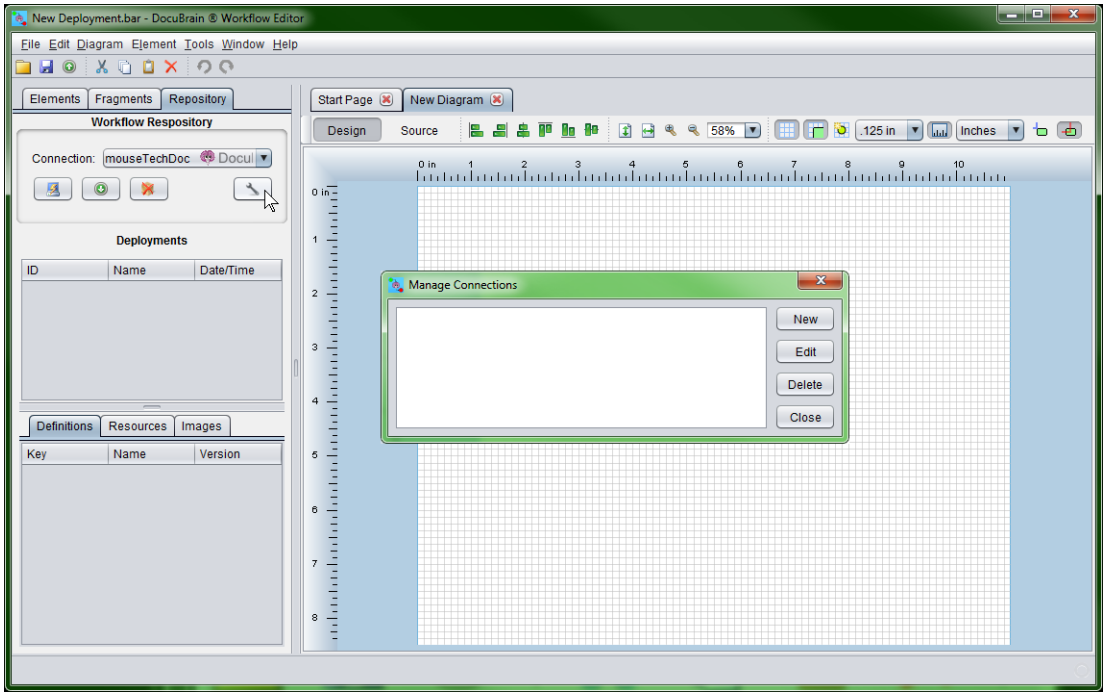
You now have a complete process that will execute and take one minute to complete. Go ahead and save this deployment; we will need it in our next exercise. On the File menu, click Save Deployment or click the save button on the toolbar. Find a location on your computer to save the deployment, give it a name and click the Save button.

You can use this deployment in the next section when we connect to a workflow repository and upload it.

4.3. *Connecting to a Workflow Engine Repository*

The DocuBrain Workflow Editor has the ability to connect directly to a TechDoc workflow engine to deploy workflow processes or download and edit existing processes.

Let's start by clicking the Repository tab on the Left Panel and then clicking the manage connections button .



On the Manage Connections dialog, click the New button. You will now be looking at the Create Connection dialog.

- Enter a name for the connection in the Connection Name box.
- Select the TechDoc version this connection will target.
- Enter the host name of the TechDoc instance; i.e. example.com making sure to leave off any trailing context path.
- Enter the base URL for a TechDoc connection is always */service/workflow/* and cannot be changed.

- Now enter the username and password of your TechDoc user account. If you are connecting via Single Sign On, leave both fields blank and check the "Connect using Single Sign On" checkbox. Please remember that the editor does NOT save your password and does not have the ability to do so. It is used here just to verify that the editor can make a connection to the workflow engine repository.

- Next, you'll need to decide whether or not the workflow repository is using HTTPS. If you visit the workflow engine in a web browser, you should be able to look at the beginning of the URL to see if it starts with "http://" or "https://".

- If you are required to connect to through a Proxy server, you may enter the proxy server and port number (if required) in the Proxy field. You should enter something like example.com or example.com:1234. If you do not use a proxy server, this field MUST be left blank.

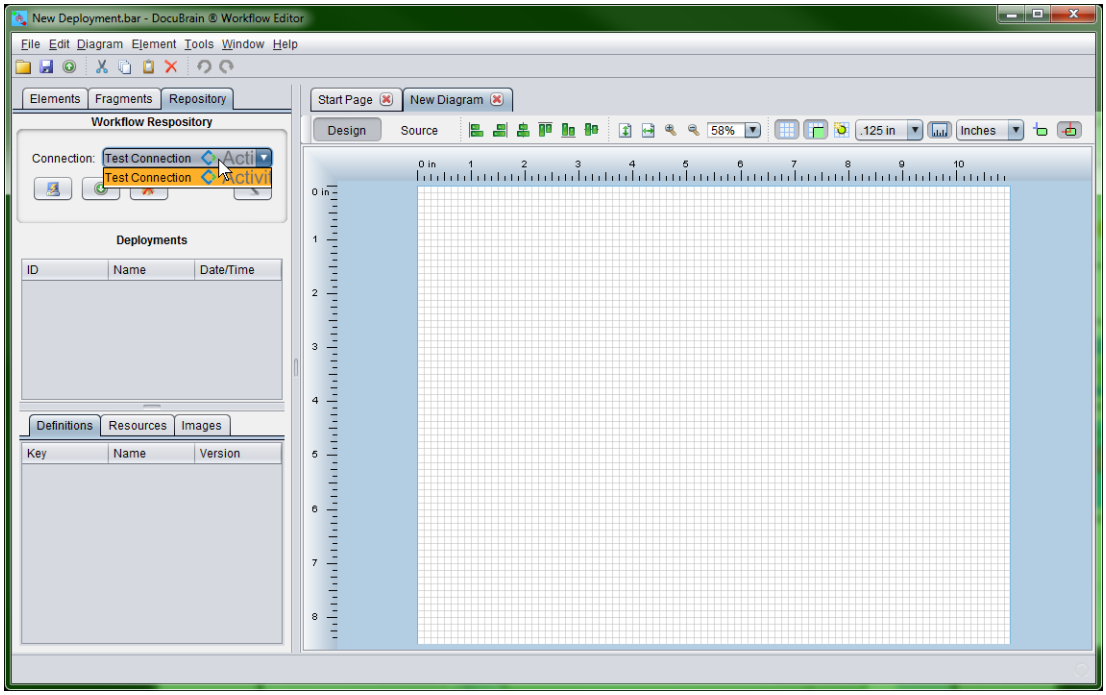
- You may optionally adjust the timeout in the Timeout field. The timeout is specified in seconds and is used for every call made to the repository. For instance, when you go to click the OK button on this dialog the editor will call the repository to make sure it can connect to it and retrieve its version. The version check call will be made and then the editor will wait to 30 seconds (the default) for the workflow repository to respond. You may adjust this field if you find the repository is commonly timing out due to a slow internet connection or high level of usage.

Once you have entered the settings for the connection to your repository, click the OK button. The editor will then test your connection settings to make sure it can connect to the repository. If your connection information is correct, the Create Connection dialog will close and you will be looking at the Manage Connections dialog again. If you instead encounter an error message, please read the message and verify your connection information is correct. Click the Close button on the Manage Connections dialog.

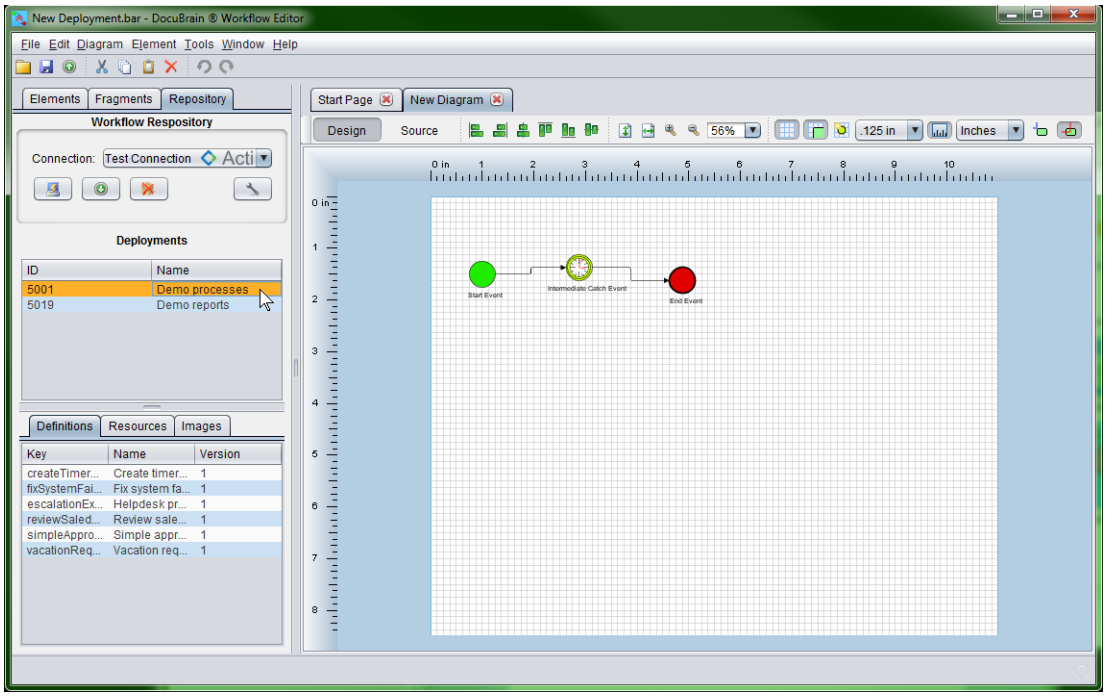
4.4. *Deploying to a Workflow Engine Repository*

In this section, we will cover uploading a deployment to a workflow engine repository. Locate a test deployment to upload or use the deployment you created in the previous section. To open the deployment in the editor, click Open Deployment on the File menu. On the Open Deployment dialog, navigate to the deployment, select it and then click the Open button. Once a deployment is open in the editor it may be uploaded to workflow engine repository.

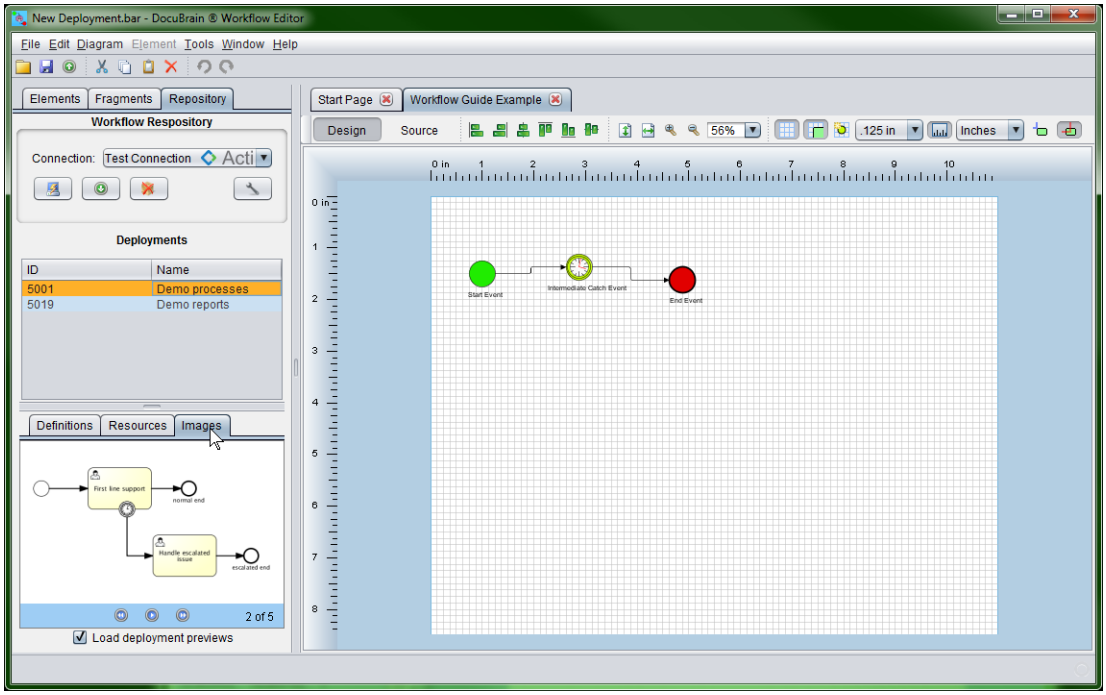
On the main editor window, click the Repository tab on the Left Panel and select your connection from the Connection drop down box.



On the Repository tab, click the connect button to connect to the workflow engine repository. When you connect, the editor will download the information for all of the workflow deployments and processes that you have access to see. If the workflow engine is does not yet contain a deployment, the list will be empty. Below is default content you will see if you connect to a TechDoc workflow repository.



Notice when you click a deployment listed in the Deployments section, you can view the process definitions it contains on the Definitions tab below it. You may also view all of the resources in the deployment; typically, this list will contain all of the bpmn20.xml files, all of the thumbnails for each process (usually PNG files) and any other miscellaneous resource files the deployment might contain. Additionally, you can click the Images tab and view the thumbnails of the processes if the "Load deployment previews" checkbox is selected.

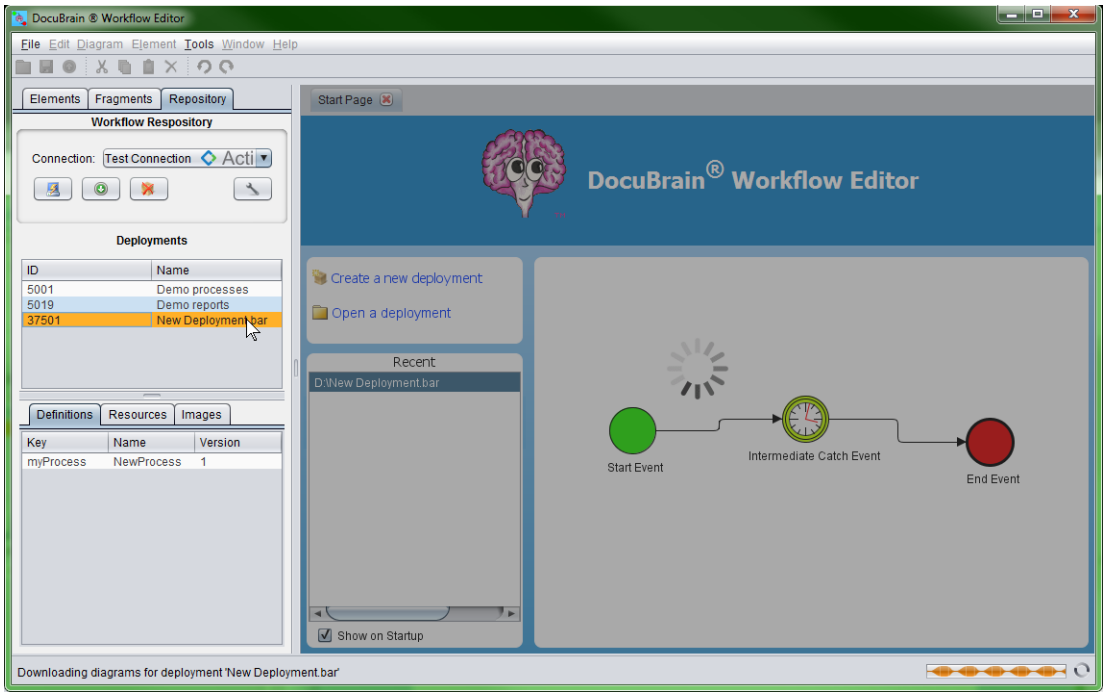


Now that we have connected to the TechDoc repository, let's upload our deployment. To do this, simply make sure your deployment is loaded in the editor and then click the upload button on the main toolbar or select Upload Deployment from the File menu. The editor will then ask you a few questions to make sure you are uploading to the repository that you mean to and ask you for a name for the deployment (the package as a whole; the name of the BAR file). Once you have answered these questions, the editor will upload your deployment on the workflow engine. You will notice that the editor then calls the workflow repository and refreshes the list of Deployments. If everything went smoothly and your deployment contained one or more legal BPMN processes, you should see your deployment in the list. If you encounter an error while uploading your deployment, you likely have an error with one of your processes. You will need to read the error message and correct the issue before the deployment can be completed.

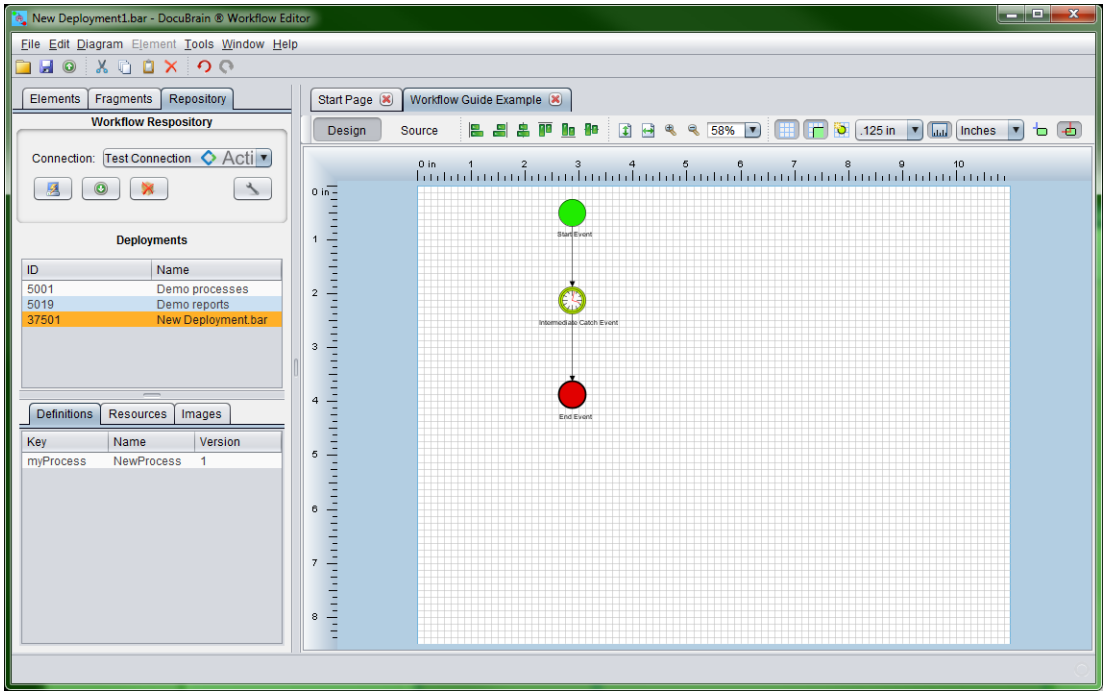
4.5. *Downloading from a Workflow Engine Repository*

The DocuBrain Workflow Editor allows you not just to upload to a workflow engine repository but also download a deployment, edit it and then upload it back. In order to download from a repository, you must have first created a connection to that workflow repository. If you have not done this, refer to the previous section Connecting to a Workflow Engine Repository.

First, we need to connect to the repository that we wish to download from. Click the Repository tab on the Left Panel, select a connection and then click the connect button . After the editor connects to the repository, it will return a list of the deployments in the repository that you have access to. On the Deployments list, select the deployment that you wish to download and then double click it or click the download button .



Once the deployment download has completed, it will be loaded into the editor where you can then edit any of the diagrams it contains. For this example, we will use our example deployment that we created in a previous section. We've downloaded the deployment back from TechDoc and we've moved the elements around so that they are now oriented vertically.



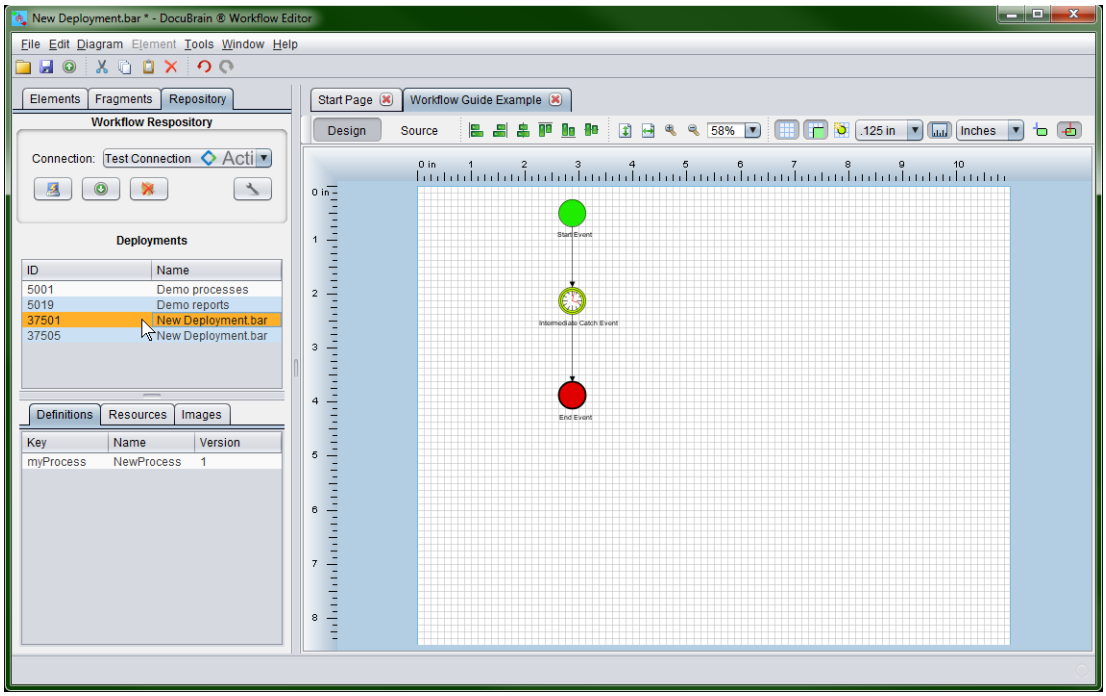
We have completed the changes we wanted to make, so we have gone ahead and saved it locally. You do not have to save a deployment locally that you have downloaded and modified from a repository. You are free to pull down deployments, make changes, and then send them right back. You will notice that the original name of the deployment was "*New Deployment.bar*" and the second modified version that we saved locally is named "*New Deployment1.bar*". We have done this so that we can maintain both the original and the second version locally. Now we will upload this deployment to TechDoc. We have a couple options:

- We can delete the original deployment from the Deployments list on the Left Panel and upload this deployment keeping the deployment name the same. This will get rid of the old deployment and this deployment will become the new version 1 of the deployment.

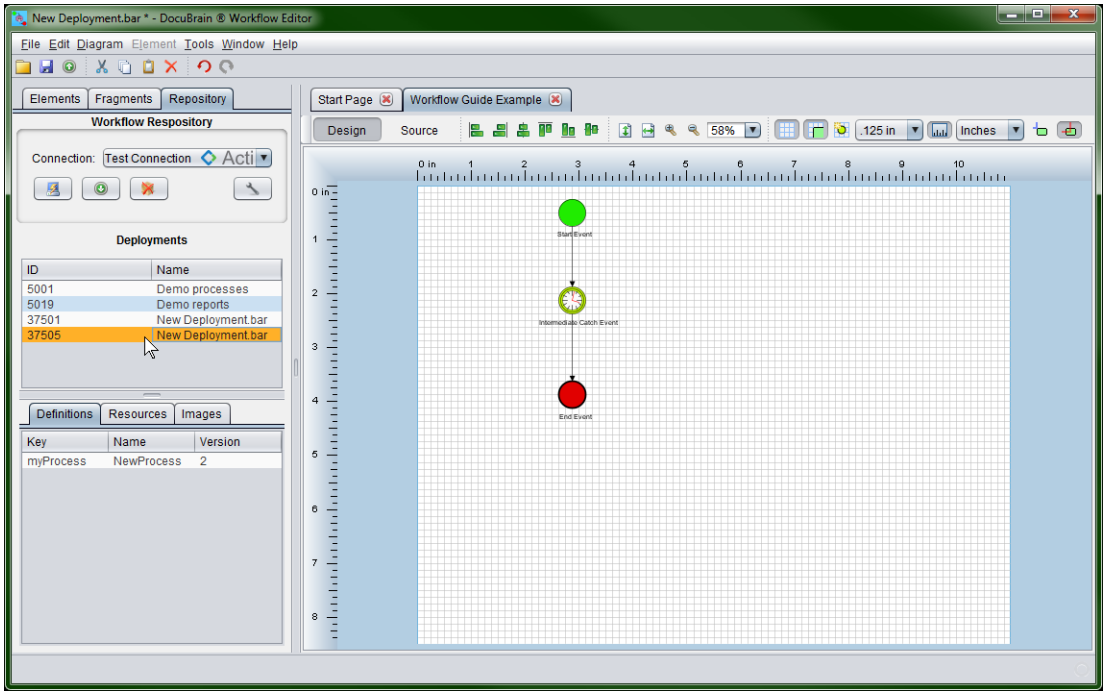
- For the second option, we can leave the original deployment untouched in the repository and upload this deployment leaving the original version 1 and creating this modified version as version 2.

You will have to decide what is right in your case. Some people choose to wait until all running instances of a deployment have completed and then delete the original and simply replace it with the new one. However, many environments do not allow for this because they have such a large usage of workflow and cannot interrupt or have down time. In a case like this, it is most appropriate to upload a new "version 2" of your deployment. The workflow engine should then begin using the new version 2 deployment automatically. Over time, all of the instances of the original (version 1) deployment should complete. After there are no more running processes, you may delete the original so that you are left with only the new modified deployment that you wish to keep.

For this example, we will go ahead and upload the modified example deployment leaving the original deployment in TechDoc. Because we have not changed the name on the process, we will end up with a version 1 (the original) and a version 2 (the modified process). This is what you will see:



If you look at the Deployments list on the Left Panel, you will see that we have selected the first *"New Deployment.bar"*. If you look at the Definitions list below it, you will see our process name *"NewProcess"* with a Version number of 1.



If you click the new deployment in the list, you'll notice that because the process contains had the same name *"NewProcess"*, the process has been deemed version 2.

4.6. *Multi-Process Deployments*

Another important feature of the DocuBrain Workflow Editor is its support for deployments that contain more than one workflow process (diagram). While there are few editors out there that have this ability, most workflow engines do support it. Multi-diagram deployments are very nice when you want to package a bunch of processes that all have something in common. For example, we package all of our TechDoc workflow tutorial processes in a single deployment. This way a user can deploy the single tutorial package and have access to all of the processes. Later on, when the tutorial processes are no longer needed the single deployment can be removed.

Additional processes can be added to a deployment, by simply opening a deployment and then selecting Add Diagram from the File menu. This will add a new empty diagram to the currently open deployment. If needed, a diagram can be removed from a deployment by clicking its tab, and then clicking the  button on the tab or selecting Remove Diagram from the File menu.

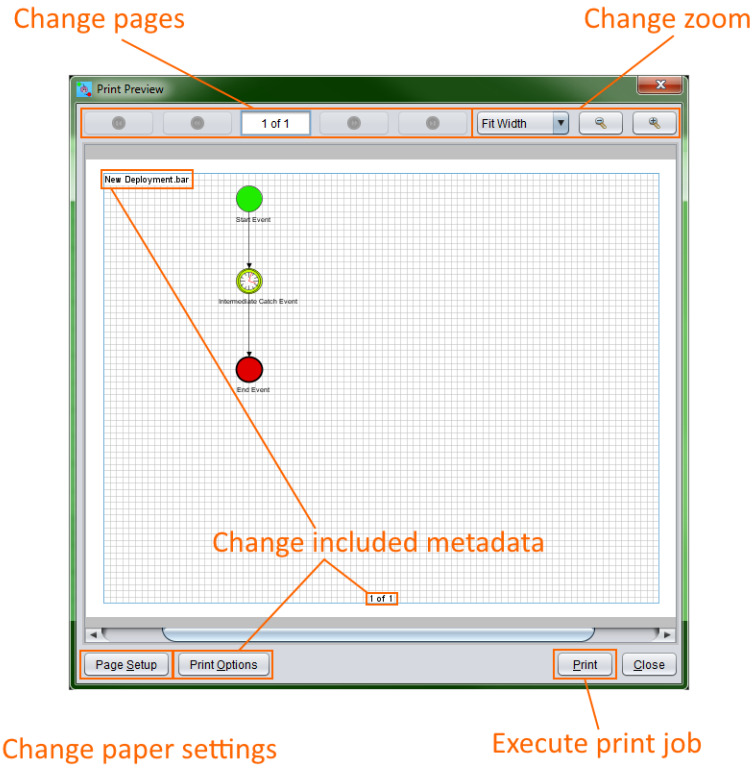
4.7. *Importing and Exporting Diagrams*

In addition to multi-process deployments, diagram can be imported or exported to and from a deployment. Currently, the DocuBrain Workflow Editor supports importing: *.bpmn*, *.bpmn20.xml*, and *XML* files. This feature is particularly handy when you want to import individual diagrams from another deployment or import a diagram from another editor. Please keep in mind, when importing from another editor, you may have to make correction to a diagram once it has been imported if it contains customizations specific to that editor or contains illegal code. Though many BPMN process editors follow the BPMN 2.0 specification, not all editors build processes in the same manner. Additionally, not all workflow engines support the same BPMN elements and most have customized attributes they use that are not defined in the BPMN 2.0 specification.

With all of this in mind, you can import a diagram in your deployment by simply clicking Import Diagram from the File menu, selecting a BPMN file, and then clicking Open. The editor will load the BPMN file and add it as an additional diagram to your deployment. In addition, you may export a diagram from your deployment by opening the deployment, selecting the diagram that you wish to export and then selecting Export Diagram from the File menu. If you encounter any issues importing or exporting diagrams or have a need to support importing from another editor or workflow editor, please voice this on our website using the Submit Feedback action on the Help menu.

4.8. *Printing a Diagram*

The DocuBrain Workflow Editor supports printing the diagrams of a deployment. To print a diagram, open/download a deployment and then select Print from the File menu. Once Print Preview dialog displays, you can preview how the print out will look using the various controls on the dialog.

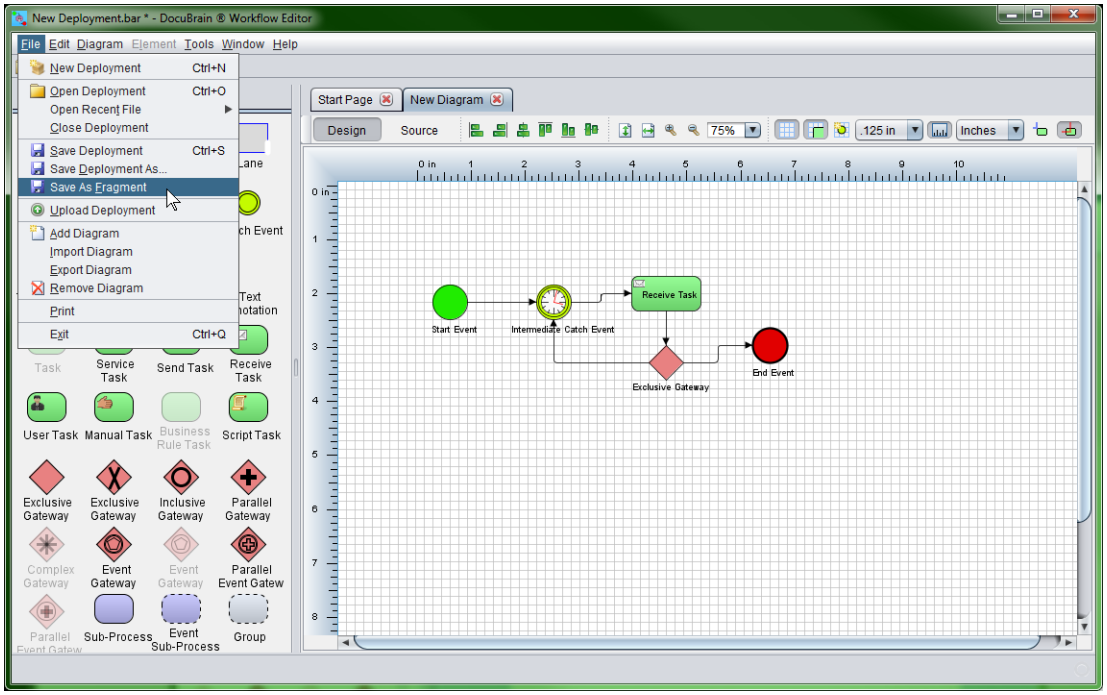


Each diagram of a deployment will be shown as a separate page in the print job. You can navigate through the pages using the buttons on the top left of the dialog. You may also zoom the diagram in and out if needed. If you need to change the print margins or paper size, click the Page Setup button. Additionally, the editor supports printing different metadata about the deployment on six places on the page, if you wish to change the title, add the date and time, etc. click the Print Options button to do so. When you are satisfied with the preview of the print job, click the Print button to begin the print job.

After clicking the Print button, you will see the print window that you usually see when printing from any other application. This print window is not a part of the DocuBrain Workflow Editor, but it generated by your operating system (Windows, Linux, Mac, etc.). Typically, you can just click the Print/OK button to begin the print job however, if you need to change the target printer, print quality, etc., you can do it there.

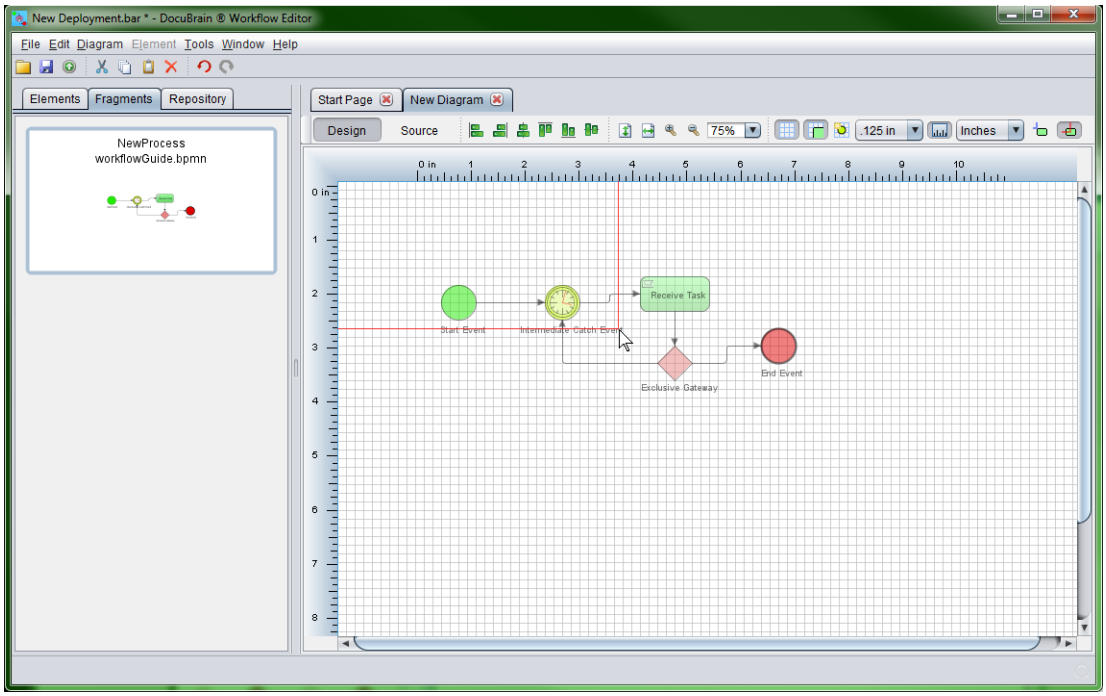
4.9. *Working with Fragments*

The DocuBrain Workflow Editor also supports working with fragments (or snippets as called by some). When you find yourself repeatedly re-creating a portion of a process, you may want to consider creating a fragment. To create a fragment, start by creating a new deployment. Now drag the elements on to the screen that you will need for your fragment. Now connect everything as needed. Once you are finished building your fragment, select "Save As Fragment" from the File menu.



Enter a name for the fragment and click Save. Consider the example above. I have a process that runs a timer once a minute, waits for a message to be received, and then decides whether to complete or loop back around. If I found myself repeatedly needing this chunk to use in many workflow processes that I create, I could save quite a bit of time creating it as a fragment so it can be reused.

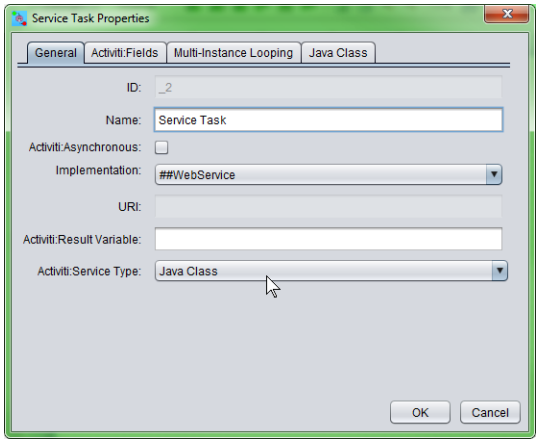
A fragment can be used any time a deployment is open. Simply click the Fragments tab on the Left Panel, then drag a fragment from the list on to your diagram like so:



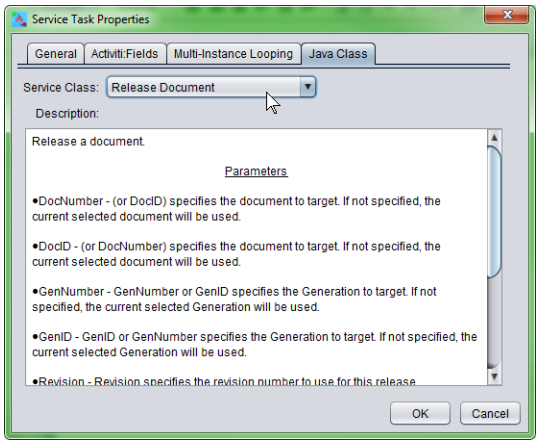
4.10. *Calling TechDoc/BPMN Service Task Operations*

The DocuBrain Workflow Editor supports calling Java class operations using a BPMN Service Task. This is the most common way to perform operations that have been specifically designed for your workflow engine. In this section, we will cover how to call TechDoc operations (also called Activities) from a workflow process.

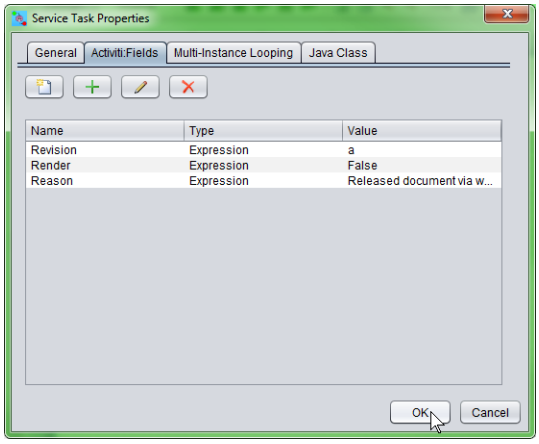
Let us build a simple process that releases a document in TechDoc when it executes. Start by dragging on to the diagram and connecting a Start Event, Service Task and End Event in that specific order. Now right click on the Service Task and then click Properties from the popup menu. You first need to make sure that your Service Type is set to Java Class.



After selecting Java Class as the Service Type, click the Java Class tab. In the Service Class drop down select Release Document. Notice the Description box displays the parameters this Java Class uses, what they expect and whether or not they are optional.



Now click the *Activiti:Fields* tab. Notice that a few fields have already been filled in for us. Go ahead and double click Revision and specify a revision. Double click Render and specify False. After you have done this, click the OK button.



You may now save this process and upload it to the workflow engine.

Did you notice that we did not enter a *DocNumber* or *DocID* variable and we also did not enter a *GenNumber* or *GenID* variable? If you do not specify a document identifier and a generation identifier (when a generation is needed), the workflow engine will try to assume the document and generation to use. For instance, say you created a trigger in the TechDoc Document Manager that ran this workflow process every time you created

a document with the Document Type Invoice. You then created an Invoice document in the TechDoc Document Manager. The workflow engine would then start this workflow process and use the document ID and generation ID of the document you just created.

Most of the time it does not make sense to explicitly set the document and generation in a workflow process since the process would only be good for a single document and generation. You could however use workflow variables to set them from other data in the process. For example, you could specify *DocID* to be *${myVariable}* to tell the workflow engine that it should find the current value for *myVariable* and set *DocID* using that variable.

5. BPMN 2.0 Constructs

This chapter covers the BPMN 2.0 constructs supported by TechDoc as well as custom extensions to the BPMN standard.

5.1. *Custom extensions*

The BPMN 2.0 standard is a good thing for all parties involved. End-users don't suffer from a vendor lock-in that comes by depending on a proprietary solution. Due to the BPMN 2.0 standard, the transition from such a big vendor solution towards TechDoc is an easy and smooth path.

The downside of a standard however, is the fact that it is always the result of many discussions and compromises between different companies (and often visions). As a developer reading the BPMN 2.0 XML of a process definition, sometimes it feels like certain constructs or way to do things are too cumbersome. Since TechDoc puts ease of development as a top-priority, we introduced something called the TechDoc BPMN extensions. These extensions are new constructs or ways to simplify certain constructs that are not in the BPMN 2.0 specification.

Although the BPMN 2.0 specification clearly states that it was made for custom extension, we make sure that:

The prerequisite of such a custom extension is that there always must be a simple transformation to the standard way of doing things. So, when you decide to use a custom extension, you don't have to be afraid that there is no way back.

When using a custom extension, this is always clearly indicated by giving the new XML element, attribute, etc. the docubrain: namespace prefix.

So, whether you want to use a custom extension or not, is completely up to you. Several factors will influence this decision (graphical editor usage, company policy, etc.). We only provide them since we believe that some points in the standard can be done simpler or more efficient. Feel free to give us (positive and/or negative) feedback on our extensions, or to post new ideas for custom extensions. Who knows, someday your idea might pop up in the specification!

5.2. *Events*

Events are used to model something that happens during the lifetime process. Events are always visualized as a circle. In BPMN 2.0, there exist two main event categories: catching or throwing event.

Catching: when process execution arrives in the event, it will wait for a trigger to happen. The type of trigger is defined by the inner icon or the type declaration in the XML. Catching events are visually differentiated from a throwing event by the inner icon that is not filled (i.e., it is white).

Throwing: when process execution arrives in the event, a trigger is fired. The type of trigger is defined by the inner icon or the type declaration in the XML. Throwing events are visually differentiated from a catching event by the inner icon that is filled with black.

5.2.1. Event Definitions

Event definitions define the semantics of an event. Without an event definition, an event "does nothing special". For instance, a start event without an event definition does not specify what exactly starts the process. If we add an event definition to the start event (like for instance a timer event definition) we declare what "type" of event starts the process (in the case of a timer event definition the fact that a certain point in time is reached).

5.2.2. Timer Event Definitions

Timer events are events which are triggered by a defined timer. They can be used as a start event, intermediate event or boundary event. The behavior of the time event depends on the business calendar used. Every timer event has a default business calendar, but the business calendar can also be defined on the timer event definition.

```
0  <timerEventDefinition docubrain:businessCalendarName="custom">
1     ...
2  </timerEventDefinition>
```

Where businessCalendarName points to business calendar in process engine configuration. When business calendar is omitted default business calendars are used.

Timer definition must have exactly one element from the following:

- **timeDate**: This format specifies a fixed date, in the ISO 8601 format, when trigger will be fired. Example:

```
1  <timerEventDefinition>
2     <timeDate>2022-03-11T12:13:14</timeDate>
3  </timerEventDefinition>
```

- **timeDuration**: To specify how long the timer should run before it is fired, a timeDuration can be specified as sub-element of timerEventDefinition. The

format used is the ISO 8601 format (as required by the BPMN 2.0 specification). Example (interval lasting 10 days):

```
<timerEventDefinition>
  <timeDuration>P10D</timeDuration>
</timerEventDefinition>
```

- **timeCycle**: Specifies a repeating interval, which can be useful for starting a process periodically, or for sending multiple reminders for overdue user task. The time cycle element can be in two formats. First is the format of recurring time duration, as specified by ISO 8601 standard. Example (3 repeating intervals, each lasting 10 hours)

There is also the possibility to specify the endDate as an optional attribute on the timeCycle or in the end of the time expression as follows: R3/PT10H/${EndDate}. When the endDate is reached, the application will stop creating other jobs for this task. It accepts as value either static values ISO 8601 standard for example "2022-02-25T16:42:11+00:00" or variables ${EndDate}

```
<timerEventDefinition>
  <timeCycle docubrain:endDate="2022-02-25T16:42:11+00:00">R3/PT10H</timeCycle>
</timerEventDefinition>
```

```
<timerEventDefinition>
  <timeCycle>R3/PT10H/${EndDate}</timeCycle>
</timerEventDefinition>
```

If both are specified then the endDate specified as attribute will be used by the system.

Currently only the BoundaryTimerEvents and CatchTimerEvent supports EndDate functionality.

5.2.3. Error Event Definitions

Important note: a BPMN error is NOT the same as a Java exception. In fact, the two have nothing in common. BPMN error events are a way of modeling business exceptions. Java exceptions are handled in their own specific way.

```
<endEvent id="myErrorEndEvent">
  <errorEventDefinition errorRef="myError" />
</endEvent>
```

5.2.4. Signal Event Definitions

Signal events are events which reference a named signal. A signal is an event of the global scope (broadcast semantics) and is delivered to all active handlers (waiting process instances/catching signal events).

A signal event definition is declared using the signalEventDefinition element. The attribute signalRef references a signal element declared as a child element of the definitions root element. The following is an excerpt of a process where a signal event is thrown and caught by intermediate events.

```
1   <definitions... >
2       <!-- declaration of the signal -->
3       <signal id="alertSignal" name="alert" />
4
5       <process id="catchSignal">
6               <intermediateThrowEvent id="throwSignalEvent" name="Alert">
7                       <!-- signal event definition -->
8                       <signalEventDefinition signalRef="alertSignal" />
9               </intermediateThrowEvent>
10              ...
11              <intermediateCatchEvent id="catchSignalEvent" name="On Alert">
12                      <!-- signal event definition -->
13                      <signalEventDefinition signalRef="alertSignal" />
14              </intermediateCatchEvent>
15              ...
16      </process>
17  </definitions>
```

The signalEventDefinitions reference the same signal element.

Throwing a Signal Event

A signal can be thrown by a process instance and either received globally by all subscribed handlers in the entire process engine or by a specific execution.

Catching a Signal Event

A signal event can be caught by an intermediate catch signal event or a signal boundary event.

Signal event scope

By default, signals are broadcast process engine wide. This means that you can throw a signal event in a process instance, and other process instances with different process definitions can react on the occurrence of this event.

However, sometimes it is wanted to react to a signal event only within the same process instance. A use case for example is a synchronization mechanism in the process instance, if two or more activities are mutually exclusive.

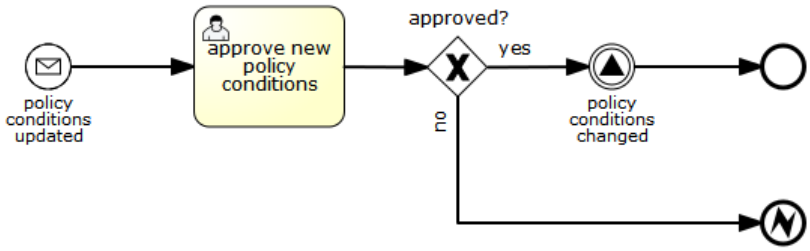
To restrict the scope of the signal event, add the (non-BPMN 2.0 standard!) scope attribute to the signal event definition:

```
1   <signal id="alertSignal" name="alert" docubrain:scope="processInstance"/>
```

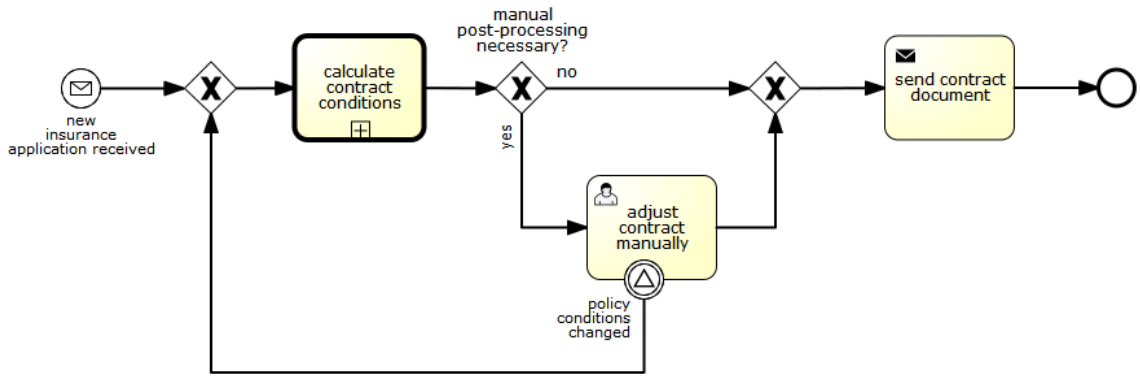
The default value for this is attribute is "global".

Signal Event example(s)

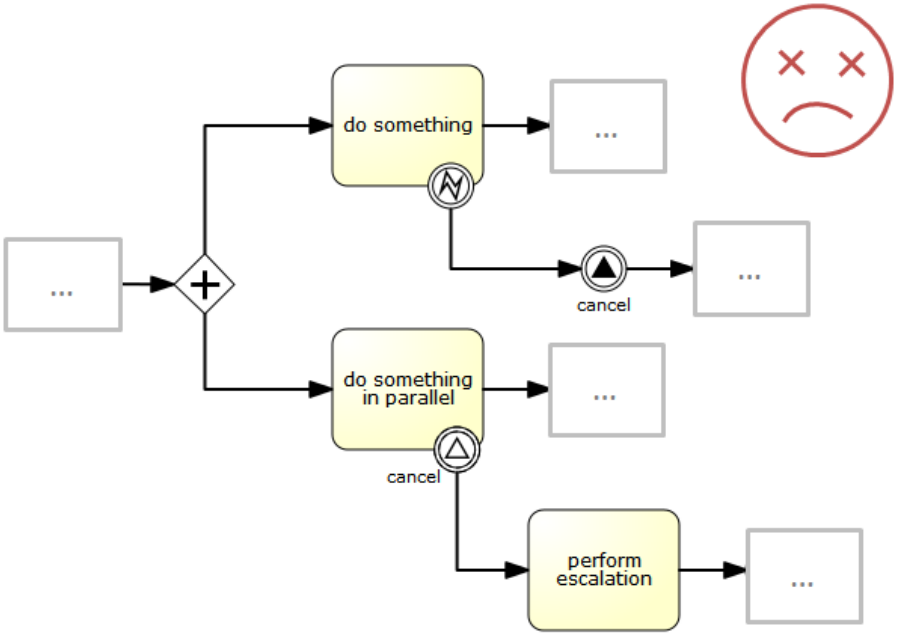
The following is an example of two separate processes communicating using signals. The first process is started if an insurance policy is updated or changed. After the changes have been reviewed by a human participant, a signal event is thrown, signaling that a policy has changed:



This event can now be caught by all process instances which are interested. The following is an example of a process subscribing to the event.



Note: it is important to understand that a signal event is broadcast to **all** active handlers. This means in the case of the example given above, that all instances of the process catching the signal would receive the event. In this case this is what we want. However, there are also situations where the broadcast behavior is unintended. Consider the following process:



The pattern described in the process above is not supported by BPMN. The idea is that the error thrown while performing the "do something" task is caught by the boundary error event and would be propagated to the parallel path of execution using the signal throw event and then interrupt the "do something in parallel" task. So far, TechDoc would perform as expected. The signal would be propagated to the catching boundary event and interrupt the task. However, due to the broadcast semantics of the signal, it would also be propagated to all other process instances which have subscribed to the signal event. In this case, this might not be what we want.

Note: the signal event does not perform any kind of correlation to a specific process instance. On the contrary, it is broadcast to all process instances. If a signal event should be kept local to the process, the scope attribute can be added to the signal event and set to processInstance.

5.2.5. Message Event Definitions

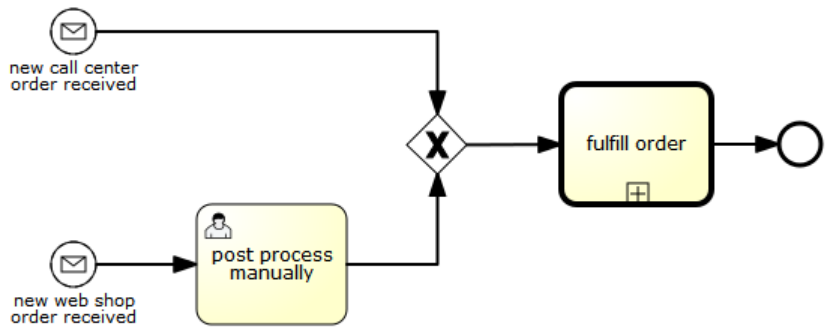
Message events are events which reference a named message. A message has a name and a payload. Unlike a signal, a message event is always directed at a single receiver.

A message event definition is declared using the messageEventDefinition element. The attribute messageRef references a message element declared as a child element of the definitions root element. The following is an excerpt of a process where two message events are declared and referenced by a start event and an intermediate catching message event.

```
1  <definitions id="definitions"
2    xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
3    targetNamespace="Examples"
4    xmlns:tns="Examples">
5
6    <message id="newInvoice" name="newInvoiceMessage" />
7    <message id="payment" name="paymentMessage" />
8
9    <process id="invoiceProcess">
10
11     <startEvent id="messageStart" >
12       <messageEventDefinition messageRef="newInvoice" />
13     </startEvent>
14     ...
15     <intermediateCatchEvent id="paymentEvt" >
16       <messageEventDefinition messageRef="payment" />
17     </intermediateCatchEvent>
18     ...
19   </process>
20
21 </definitions>
```

Message Event example(s)

The following is an example of a process which can be started using two different messages:



This is useful if the process needs alternative ways to react to different start events but eventually continues in a uniform way.

5.2.6. Start Events

A start event indicates where a process starts. The type of start event (process starts on arrival of message, on specific time intervals, etc.), defining how the process is started is shown as a small icon in the visual representation of the event. In the XML representation, the type is given by the declaration of a sub-element.

Start events **are always catching**: conceptually the event is (at any time) waiting until a certain trigger happens.

5.2.7. Timer Start Event

Description

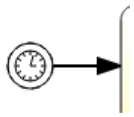
A timer start event is used to create a process instance at given time. It can be used both for processes which should start only once and for processes that should start in specific time intervals.

Note: a subprocess cannot have a timer start event.

Note: when a new version of a process with a start timer event is deployed, the job corresponding with the previous timer will be removed. The reasoning is that normally it is not wanted to keep automatically starting new process instances of this old version of the process.

Graphical notation

A none start event is visualized as a circle with clock inner icon.



XML representation

The XML representation of a timer start event is the normal start event declaration, with timer definition sub-element. Please refer to timer definitions for configuration details.

Example: process will start 4 times, in 5-minute intervals, starting on 11th march 2022, 12:13

```
<startEvent id="theStart">
  <timerEventDefinition>
    <timeCycle>R4/2022-03-11T12:13/PT5M</timeCycle>
</timerEventDefinition>
</startEvent>
```

Example: process will start once, on selected date

```
<startEvent id="theStart">
  <timerEventDefinition>
    <timeDate>2022-03-11T12:13:14</timeDate>
  </timerEventDefinition>
</startEvent>
```

5.2.8. Message Start Event

Description

A message start event can be used to start a process instance using a named message. This effectively allows us to select the right start event from a set of alternative start events using the message name.

When **deploying** a process definition with one or more message start events, the following considerations apply:

- The name of the message start event must be unique across a given process definition. A process definition must not have multiple message start events with the same name. TechDoc will throw an exception upon deployment of a process definition with two or more message start events that reference the same message name.

- The name of the message start event must be unique across all deployed process definitions. TechDoc will throw an exception upon deployment of a process definition with one or more message start events that reference a message with the same name as a message start event already deployed by a different process definition.

- Process versioning: Upon deployment of a new version of a process definition, the message subscriptions of the previous version are cancelled. This is also true for message events that are not present in the new version.

Graphical notation

A message start event is visualized as a circle with a message event symbol. The symbol is unfilled, to visualize the catching (receiving) behavior.

XML representation

The XML representation of a message start event is the normal start event declaration with a messageEventDefinition child-element:

```
1   <definitions id="definitions"
2     xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
3     targetNamespace="Examples"
4     xmlns:tns="Examples">
5
6     <message id="newInvoice" name="newInvoiceMessage" />
7
8     <process id="invoiceProcess">
9
10      <startEvent id="messageStart" >
11        <messageEventDefinition messageRef="tns:newInvoice" />
12      </startEvent>
13      ...
14    </process>
15
16  </definitions>
```

5.2.9. Signal Start Events

Description

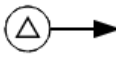
A signal start event can be used to start a process instance using a named signal. The signal can be fired from within a process instance using the intermediary signal throw event. All process definitions that have a signal start event with the same name will be started.

Note: in both cases, it is also possible to choose between a synchronous and asynchronous starting of the process instances.

The signalName that must be passed to TechDoc is the name given in the name attribute of the signal element referenced by the signalRef attribute of the signalEventDefinition.

Graphical notation

A signal start event is visualized as a circle with a signal event symbol. The symbol is unfilled, to visualize the catching (receiving) behavior.



XML representation

The XML representation of a signal start event is the normal start event declaration with a signalEventDefinition child-element:

```
1   <signal id="theSignal" name="The Signal" />
2
3   <process id="processWithSignalStart1">
4     <startEvent id="theStart">
5       <signalEventDefinition id="theSignalEventDefinition" signalRef="theSignal"  />
6     </startEvent>
7     <sequenceFlow id="flow1" sourceRef="theStart" targetRef="theTask" />
8     <userTask id="theTask" name="Task in process A" />
9     <sequenceFlow id="flow2" sourceRef="theTask" targetRef="theEnd" />
10        <endEvent id="theEnd" />
11  </process>
```

5.2.10. Error Start Event

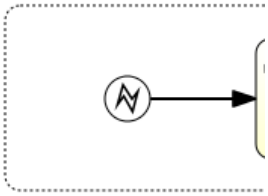
Description

An error start event can be used to trigger an Event Sub-Process. An error start event cannot be used for starting a process instance.

Note: an error start event is always interrupting.

Graphical notation

An error start event is visualized as a circle with an error event symbol. The symbol is unfilled, to visualize the catching (receiving) behavior.



XML representation

The XML representation of an error start event is the normal start event declaration with an errorEventDefinition child-element:

```
1   <startEvent id="messageStart" >
2         <errorEventDefinition errorRef="someError" />
3   </startEvent>
```

5.2.11. End Events

An end event signifies the end (of a path) of a (sub)process. An end event is always throwing. This means that when a process execution arrives in the end event, a result is thrown. The type of result is depicted by the inner black icon of the event. In the XML representation, the type is given by the declaration of a sub-element.

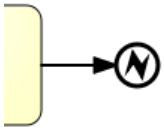
5.2.12. Error End Event

Description

When a process execution arrives in an error end event, the current path of execution is ended and an error is thrown. This error can be caught by a matching intermediate boundary error event. If there is no matching boundary error event found, an exception will be thrown.

Graphical notation

An error end event is visualized as a typical end event (circle with thick border), with the error icon inside. The error icon is completely black, to indicate the throwing semantics.



XML representation

An error end event is represented as an end event with an errorEventDefinition child element.

```
1   <endEvent id="myErrorEndEvent">
2     <errorEventDefinition errorRef="myError" />
3   </endEvent>
```

The errorRef attribute can reference an error element that is defined outside the process:

```
1   <error id="myError" errorCode="123" />
2   ...
3   <process id="myProcess">
4   ...
```

The errorCode of the error will be used to find the matching catching boundary error event. If the errorRef does not match any defined error, then the errorRef is used as a shortcut for the errorCode. This is a TechDoc specific shortcut. More concretely, following snippets are equivalent in functionality.

```
1   <error id="myError" errorCode="error123" />
2   ...
3   <process id="myProcess">
4   ...
5     <endEvent id="myErrorEndEvent">
6       <errorEventDefinition errorRef="myError" />
7     </endEvent>
8   ...
```

is equivalent with

```
1   <endEvent id="myErrorEndEvent">
2     <errorEventDefinition errorRef="error123" />
3   </endEvent>
```

Note that the errorRef must comply with the BPMN 2.0 schema, and must be a valid QName.

5.2.13. Terminate End Event

Description

When a terminate end event is reached, the current process instance or sub-process will be terminated. Conceptually, when an execution arrives in a terminate end event, the first scope (process or sub-process) will be determined and ended. Note that in BPMN 2.0, a sub-process can be an embedded sub-process, call activity, event sub-process or transaction sub-process. This rule applies in general: when for example there is a multi-instance call activity or embedded subprocess, only that instance will be ended, the other instances and the process instance are not affected.

There is an optional attribute terminateAll that can be added. When true, regardless of the placement of the terminate end event in the process definition and regardless of being in a sub-process (even nested), the (root) process instance will be terminated.

Graphical notation

A cancel end event visualized as a typical end event (circle with thick outline), with a full black circle inside.

XML representation
A terminate end event is represented as an end event, with a terminateEventDefinition child element.

Note that the terminateAll attribute is optional (and false by default).

```
1  <endEvent id="myEndEvent >
2   <terminateEventDefinition
   docubrain:terminateAll="true"></terminateEventDefinition>
3  </endEvent>
```

5.2.14. Boundary Events

Boundary events are catching events that are attached to an activity (a boundary event can never be throwing). This means that while the activity is running, the event is listening for a certain type of trigger. When the event is caught, the activity is interrupted and the sequence flow going out of the event is followed.

All boundary events are defined in the same way:

```
1  <boundaryEvent id="myBoundaryEvent" attachedToRef="theActivity">
2      <XXXEventDefinition/>
3  </boundaryEvent>
```

A boundary event is defined with

- A unique identifier (process-wide)
- A reference to the activity to which the event is attached through the attachedToRef attribute. Note that a boundary event is defined on the same level as the activities to which they are attached (i.e., no inclusion of the boundary event inside the activity).
- An XML sub-element of the form XXXEventDefinition (e.g. TimerEventDefinition, ErrorEventDefinition, etc.) defining the type of the boundary event. See the specific boundary event types for more details.

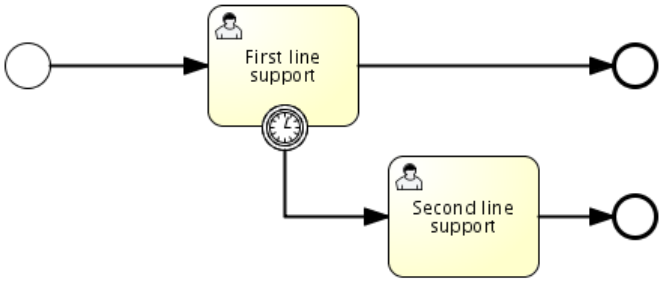
5.2.15. Timer Boundary Event

Description

A timer boundary event acts as a stopwatch and alarm clock. When an execution arrives in the activity where the boundary event is attached to, a timer is started. When the timer fires (e.g., after a specified interval), the activity is interrupted and the boundary event is followed.

Graphical Notation

A timer boundary event is visualized as a typical boundary event (i.e., circle on the border), with the timer icon on the inside.



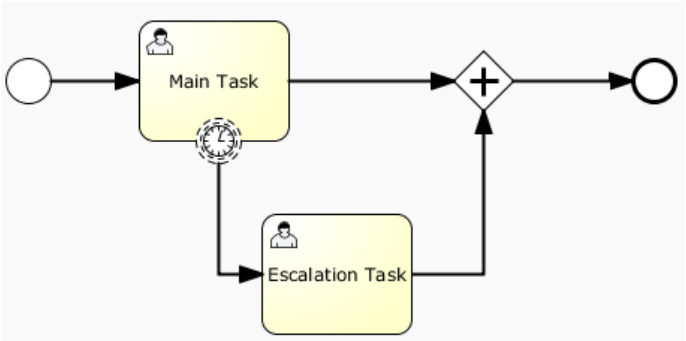
XML Representation

A timer boundary event is defined as a regular boundary event. The specific type sub-element is in this case a timerEventDefinition element.

```
1  <boundaryEvent id="escalationTimer" cancelActivity="true"
   attachedToRef="firstLineSupport">
2    <timerEventDefinition>
3      <timeDuration>PT4H</timeDuration>
4    </timerEventDefinition>
5  </boundaryEvent>
```

Please refer to timer event definition for details on timer configuration.

In the graphical representation, the line of the circle is dotted as you can see in this example below:



A typical use case is sending an escalation email additionally but not interrupt the normal process flow.

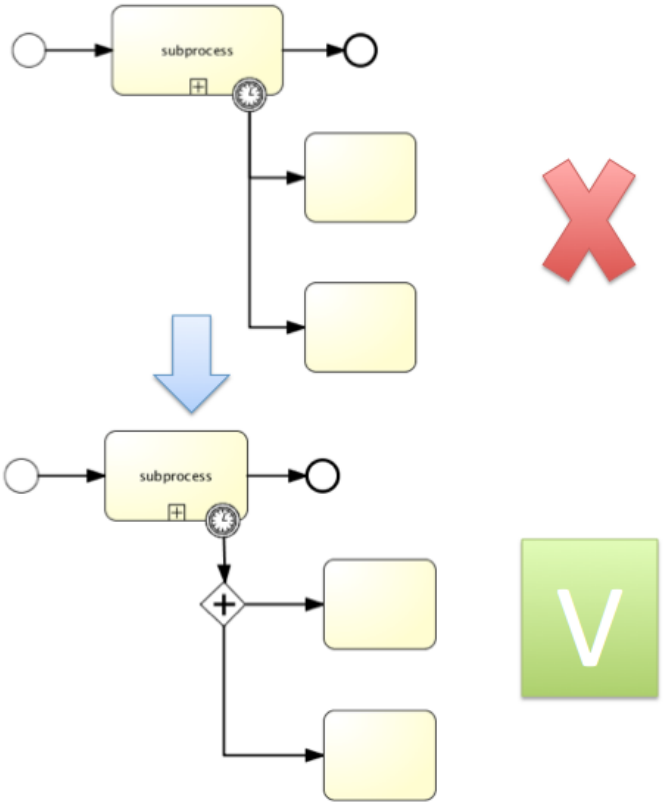
Since BPMN 2.0 there is a difference between the interrupting and non-interrupting timer event. Interrupting is the default. The non-interrupting event leads to the original activity and is not interrupted and the activity stays there. Instead, additional executions are created and they are sent over the outgoing transition of the event. In the XML representation, the cancelActivity attribute is set to false:

```
1  <boundaryEvent id="escalationTimer" cancelActivity="false"
   attachedToRef="firstLineSupport"/>
```

Known issue with boundary events

There is a known issue regarding concurrency when using boundary events of any type. Currently, it is not possible to have multiple outgoing sequence flow attached to a

boundary event. A solution to this problem is to use one outgoing sequence flow that goes to a parallel gateway.



5.2.16. Error Boundary Event

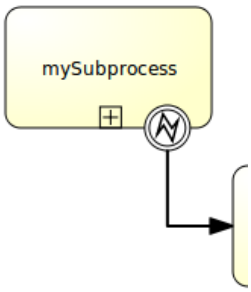
Description
An intermediate catching error on the boundary of an activity, or boundary error event for short, catches errors that are thrown within the scope of the activity on which it is defined.

Defining a boundary error event makes the most sense on an embedded subprocess, or a call activity, as a subprocess creates a scope for all activities inside the subprocess. Errors are thrown by error end events. Such an error will propagate its parent scopes upwards until a scope is found on which a boundary error event is defined that matches the error event definition.

When an error event is caught, the activity on which the boundary event is defined is destroyed, also destroying all current executions within (e.g., concurrent activities, nested subprocesses, etc.). Process execution continues following the outgoing sequence flow of the boundary event.

Graphical notation

A boundary error event is visualized as a typical intermediate event (circle with smaller circle inside) on the boundary, with the error icon inside. The error icon is white, to indicate the catch semantics.



Xml representation

A boundary error event is defined as a typical boundary event:

```
1  <boundaryEvent id="catchError" attachedToRef="mySubProcess">
2    <errorEventDefinition errorRef="myError"/>
3  </boundaryEvent>
```

As with the error end event, the errorRef references an error defined outside the process element:

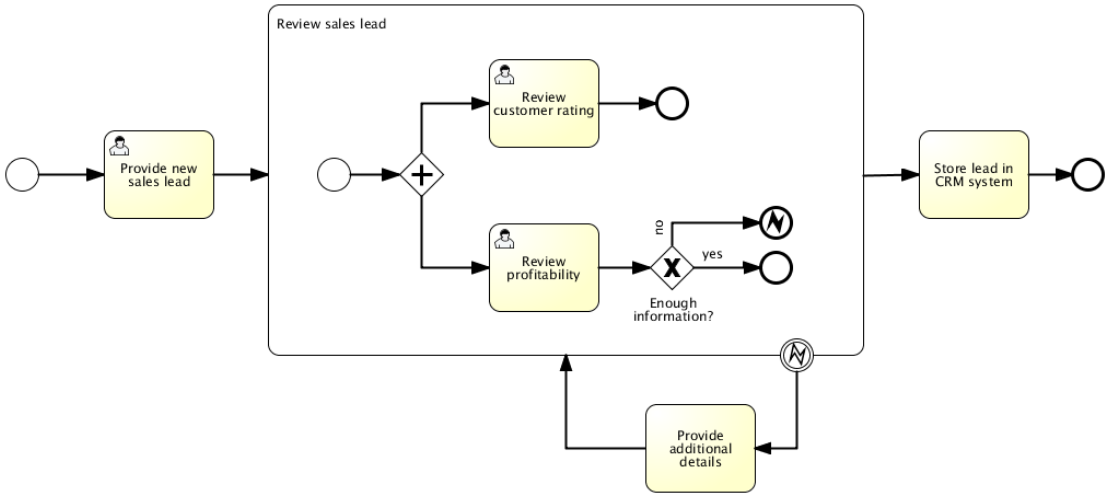
```
1  <error id="myError" errorCode="123" />
2  ...
3  <process id="myProcess">
4  ...
```

The errorCode is used to match the errors that are caught:

- If errorRef is omitted, the boundary error event will catch any error event, regardless of the errorCode of the error.
- In case an errorRef is provided and it references an existing error, the boundary event will only catch errors with the same error code.
- In case an errorRef is provided, but no error is defined in the BPMN 2.0 file, then the errorRef is used as errorCode (similar for with error end events).

The following example process shows how an error end event can be used. When the 'Review profitability' user task is completed by stating that not enough information is provided, an error is thrown. When this error is caught on the boundary of the

subprocess, all active activities within the 'Review sales lead' subprocess are destroyed (even if 'Review customer rating' was not yet completed), and the 'Provide additional details' user task is created.



5.2.17. Signal Boundary Event

Description

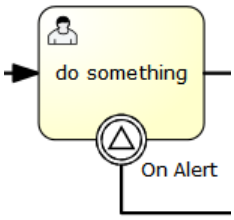
An attached intermediate catching signal on the boundary of an activity, or boundary signal event for short, catches signals with the same signal name as the referenced signal definition.

Note: contrary to other events like the boundary error event, a boundary signal event does not only catch signal events thrown from the scope it is attached to. A signal event has global scope (broadcast semantics) meaning that the signal can be thrown from any place, even from a different process instance.

Note: contrary to other events like an error event, a signal is not consumed if it is caught. If you have two active signal boundary events catching the same signal event, both boundary events are triggered, even if they are part of different process instances.

Graphical notation

A boundary signal event is visualized as a typical intermediate event (Circle with smaller circle inside) on the boundary, with the signal icon inside. The signal icon is white (unfilled), to indicate the catch semantics.



XML representation

A boundary signal event is defined as a typical boundary event:

```
1   <boundaryEvent id="boundary" attachedToRef="task" cancelActivity="true">
2           <signalEventDefinition signalRef="alertSignal"/>
3   </boundaryEvent>
```

Example

See section on signal event definitions.

5.2.18. Message Boundary Event

Description

An attached intermediate catching message on the boundary of an activity, or boundary message event for short, catches messages with the same message name as the referenced message definition.

Graphical notation

A boundary message event is visualized as a typical intermediate event (Circle with smaller circle inside) on the boundary, with the message icon inside. The message icon is white (unfilled), to indicate the catch semantics.

Note that boundary message event can be both interrupting (right hand side) and non-interrupting (left hand side).

XML representation

A boundary message event is defined as a typical boundary event:

```
1  <boundaryEvent id="boundary" attachedToRef="task" cancelActivity="true">
2        <messageEventDefinition messageRef="newCustomerMessage"/>
3  </boundaryEvent>
```

Example

See section on message event definitions.

5.2.19. Compensation Boundary Event

Description

An attached intermediate catching compensation on the boundary of an activity or compensation boundary event for short, can be used to attach a compensation handler to an activity.

The compensation boundary event must reference a single compensation handler using a directed association.

A compensation boundary event has a different activation policy from other boundary events. Other boundary events, like for instance the signal boundary event, are activated when the activity they are attached to is started. When the activity is left, they are deactivated and the corresponding event subscription is cancelled. The compensation boundary event is different. The compensation boundary event is activated when the activity it is attached to **completes successfully**. At this point, the corresponding subscription to the compensation events is created. The subscription is removed either when a compensation event is triggered or when the corresponding process instance ends. From this, it follows:

- When compensation is triggered, the compensation handler associated with the compensation boundary event is invoked the same number of times the activity it is attached to completed successfully.

- If a compensation boundary event is attached to an activity with multiple instance characteristics, a compensation event subscription is created for each instance.

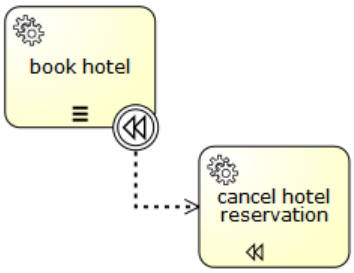
- If a compensation boundary event is attached to an activity which is contained inside a loop, a compensation event subscription is created for each time the activity is executed.

- If the process instance ends, the subscriptions to compensation events are cancelled.

Note: the compensation boundary event is not supported on embedded subprocesses.

Graphical notation

A compensation boundary event is visualized as a typical intermediate event (Circle with smaller circle inside) on the boundary, with the compensation icon inside. The compensation icon is white (unfilled), to indicate the catching semantics. In addition to a compensation boundary event, the following figure shows a compensation handler associated with the boundary event using a unidirectional association:



XML representation

A compensation boundary event is defined as a typical boundary event:

```
1  <boundaryEvent id="compensateBookHotelEvt" attachedToRef="bookHotel" >
2        <compensateEventDefinition />
3  </boundaryEvent>
4
5  <association associationDirection="One" id="a1"
   sourceRef="compensateBookHotelEvt" targetRef="undoBookHotel" />
6
7  <serviceTask id="undoBookHotel" isForCompensation="true" docubrain:class="..."
   />
```

Since the compensation boundary event is activated after the activity has completed successfully, the cancelActivity attribute is not supported.

5.2.20. Intermediate Catching Events

All intermediate catching events are defined in the same way:

```
1  <intermediateCatchEvent id="myIntermediateCatchEvent" >
```

```
2       <XXXEventDefinition/>
3   </intermediateCatchEvent>
```

An intermediate catching event is defined with

- A unique identifier (process-wide)
- An XML sub-element of the form XXXEventDefinition (e.g., TimerEventDefinition, etc.) defining the type of the intermediate catching event. See the specific catching event types for more details.

5.2.21. Timer Intermediate Catching Event

Description

A timer intermediate event acts as a stopwatch. When an execution arrives in a catching event activity, a timer is started. When the timer fires (e.g., after a specified interval), the sequence flow going out of the timer intermediate event is followed.

Graphical Notation

A timer intermediate event is visualized as an intermediate catching event, with the timer icon on the inside.

XML Representation

A timer intermediate event is defined as an intermediate catching event. The specific type sub-element in this case is a timerEventDefinition element.

```
1   <intermediateCatchEvent id="timer">
2     <timerEventDefinition>
3       <timeDuration>PT5M</timeDuration>
4     </timerEventDefinition>
5   </intermediateCatchEvent>
```

See timer event definitions for configuration details.

5.2.22. Signal Intermediate Catching Event

Description

An intermediate catching signal event catches signals with the same signal name as the referenced signal definition.

Note: contrary to other events like an error event, a signal is not consumed if it is caught. If you have two active signal boundary events catching the same signal event, both boundary events are triggered, even if they are part of different process instances.

Graphical notation

An intermediate signal catch event is visualized as a typical intermediate event (Circle with smaller circle inside), with the signal icon inside. The signal icon is white (unfilled), to indicate the catch semantics.

XML representation

A signal intermediate event is defined as an intermediate catching event. The specific type sub-element is in this case a signalEventDefinition element.

```
1  <intermediateCatchEvent id="signal">
2    <signalEventDefinition signalRef="newCustomerSignal" />
3  </intermediateCatchEvent>
```

Example

See section on signal event definitions.

5.2.23. Message Intermediate Catching Event

Description

An intermediate catching message event catches messages with a specified name.

Graphical notation

An intermediate catching message event is visualized as a typical intermediate event (Circle with smaller circle inside), with the message icon inside. The message icon is white (unfilled), to indicate the catch semantics.

XML representation

A message intermediate event is defined as an intermediate catching event. The specific type sub-element is in this case a messageEventDefinition element.

```
1  <intermediateCatchEvent id="message">
2    <messageEventDefinition signalRef="newCustomerMessage" />
3  </intermediateCatchEvent>
```

Example

See section on message event definitions.

5.2.24. Intermediate Throwing Event

All intermediate throwing events are defined in the same way:

```
1  <intermediateThrowEvent id="myIntermediateThrowEvent" >
2      <XXXEventDefinition/>
3  </intermediateThrowEvent>
```

An intermediate throwing event is defined with:

- A unique identifier (process-wide)
- An XML sub-element of the form XXXEventDefinition (e.g., signalEventDefinition, etc.) defining the type of the intermediate throwing event. See the specific throwing event types for more details.

5.2.25. Signal Intermediate Throwing Event

Description

An intermediate throwing signal event throws a signal event for a defined signal.

In TechDoc, the signal is broadcast to all active handlers (i.e., all catching signal events). Signals can be published synchronous or asynchronous.

In the default configuration, the signal is delivered synchronously. This means that the throwing process instance waits until the signal is delivered to all catching process instances. The catching process instances are also notified in the same transaction as the throwing process instance, which means that if one of the notified instances produces a technical error (throws an exception), all involved instances fail.

A signal can also be delivered asynchronously. In that case it is determined which handlers are active at the time the throwing signal event is reached. For each active handler, an asynchronous notification message (Job) is stored and delivered by the JobExecutor.

Graphical notation

An intermediate signal throw event is visualized as a typical intermediate event (Circle with smaller circle inside), with the signal icon inside. The signal icon is black (filled), to indicate the throw semantics.

XML representation

A signal intermediate event is defined as an intermediate throwing event. The specific type sub-element is in this case a signalEventDefinition element.

```
1  <intermediateThrowEvent id="signal">
2    <signalEventDefinition signalRef="newCustomerSignal" />
3  </intermediateThrowEvent>
```

An asynchronous signal event would look like this:

```
1  <intermediateThrowEvent id="signal">
2    <signalEventDefinition signalRef="newCustomerSignal" activiti:async="true" />
3  </intermediateThrowEvent>
```

Example

See section on signal event definitions.

5.2.26. Compensation Intermediate Throwing Event

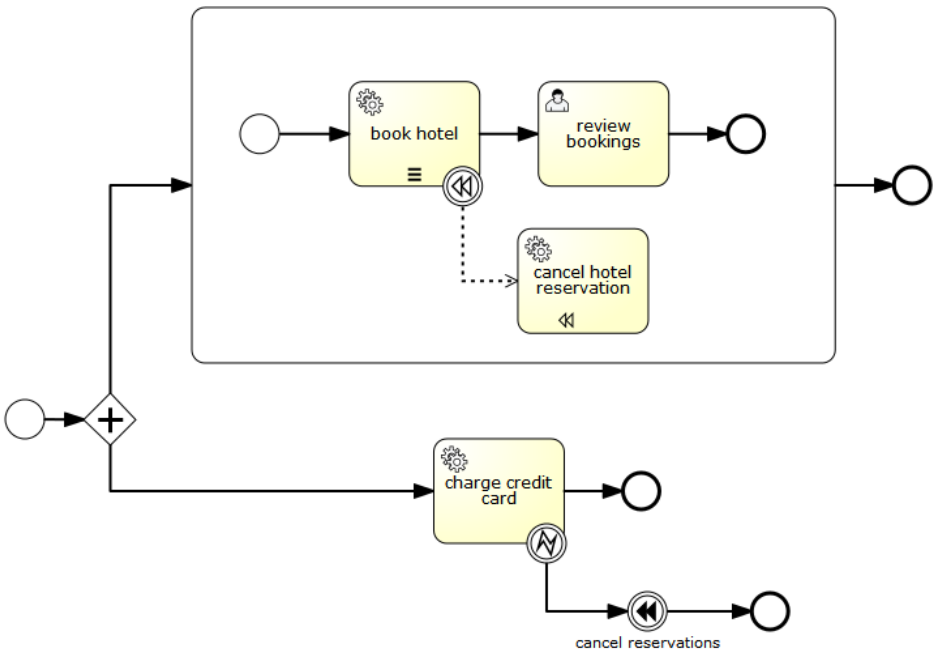
Description

An intermediate throwing compensation event can be used to trigger compensation.

Triggering compensation: Compensation can either be triggered for a designated activity or for the scope which hosts the compensation event. Compensation is performed through execution of the compensation handler associated with an activity.

- When compensation is thrown for an activity, the associated compensation handler is executed the same number of times the activity completed successfully.
- If compensation is thrown for the current scope, all activities within the current scope are compensated, which includes activities on concurrent branches.
- Compensation is triggered hierarchically: if an activity to be compensated is a subprocess, compensation is triggered for all activities contained in the subprocess. If the subprocess has nested activities, compensation is thrown recursively. However, compensation is not propagated to the "upper levels" of the process: if compensation is triggered within a subprocess, it is not propagated to activities outside of the subprocess scope. The BPMN specification states that compensation is triggered for activities at "the same level of subprocess".
- In TechDoc, compensation is performed in reverse order of execution. This means that whichever activity completed last is compensated first, etc.
- The intermediate throwing compensation event can be used to compensate transaction subprocesses which completed successfully.

Note: If compensation is thrown within a scope which contains a subprocess and the subprocess contains activities with compensation handlers, compensation is only propagated to the subprocess if it has completed successfully when compensation is thrown. If some of the activities nested inside the subprocess have completed and have attached compensation handlers, the compensation handlers are not executed if the subprocess containing these activities is not completed yet. Consider the following example:



In this process we have two concurrent executions, one executing the embedded subprocess and one executing the "charge credit card" activity. Let's assume both executions are started and the first concurrent execution is waiting for a user to complete the "review bookings" task. The second execution performs the "charge credit card" activity and an error is thrown, which causes the "cancel reservations" event to trigger compensation. At this point the parallel subprocess is not yet completed which means that the compensation event is not propagated to the subprocess and thus the "cancel hotel reservation" compensation handler is not executed. If the user task (and thus the embedded subprocess) completes before the "cancel reservations" is performed, compensation is propagated to the embedded subprocess.

Process variables: When compensating an embedded subprocess, the execution used for executing the compensation handlers has access to the local process variables of the subprocess in the state they were in when the subprocess completed execution. To achieve this, a snapshot of the process variables associated with the scope execution (execution created for executing the subprocess) is taken. Form this, a couple of implications follow:

- The compensation handler does not have access to variables added to concurrent executions created inside the subprocess scope.
- Process variables associated with executions higher up in the hierarchy, (for instance process variables associated with the process instance execution are not contained in the snapshot: the compensation handler has access to these process variables in the state they are in when compensation is thrown.

- A variable snapshot is only taken for embedded subprocesses, not for other activities.

Current limitations:

- waitForCompletion="false" is currently unsupported. When compensation is triggered using the intermediate throwing compensation event, the event is only left, after compensation completed successfully.
- Compensation itself is currently performed by concurrent executions. The concurrent executions are started in reverse order in which the compensated activities completed. Future versions of activity might include an option to perform compensation sequentially.
- Compensation is not propagated to sub process instances spawned by call activities.

Graphical notation

An intermediate compensation throw event is visualized as a typical intermediate event (Circle with smaller circle inside), with the compensation icon inside. The compensation icon is black (filled), to indicate the throw semantics.

Xml representation

A compensation intermediate event is defined as an intermediate throwing event. The specific type sub-element is in this case a compensateEventDefinition element.

```
1  <intermediateThrowEvent id="throwCompensation">
2      <compensateEventDefinition />
3  </intermediateThrowEvent>
```

In addition, the optional argument activityRef can be used to trigger compensation of a specific scope / activity:

```
1  <intermediateThrowEvent id="throwCompensation">
2      <compensateEventDefinition activityRef="bookHotel" />
3  </intermediateThrowEvent>
```

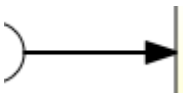
5.3. *Sequence Flow*

5.3.1. Description

A sequence flow is the connector between two elements of a process. After an element is visited during process execution, all outgoing sequence flow will be followed. This means that the default nature of BPMN 2.0 is to be parallel: two outgoing sequence flows will create two separate, parallel paths of execution.

5.3.2. Graphical Notation

A sequence flow is visualized as an arrow going from the source element towards the target element. The arrow always points towards the target.



5.3.3. XML Representation

Sequence flow need to have a process-unique id, and a reference to an existing source and target element.

```
1   <sequenceFlow id="flow1" sourceRef="theStart" targetRef="theTask" />
```

5.3.4. Conditional Sequence Flow

Description

A sequence flow can have a condition defined on it. When a BPMN 2.0 activity is left, the default behavior is to evaluate the conditions on the outgoing sequence flow. When a condition evaluates to true, that outgoing sequence flow is selected. When multiple sequence flows are selected in this way, multiple executions will be generated and the process will be continued in a parallel manner.

Note: the above holds true for BPMN 2.0 activities (and events), but not for gateways. Gateways will handle sequence flows with conditions in specific ways, depending on the gateway type.

Graphical notation

A conditional sequence flow is visualized as a regular sequence flow, with a small diamond at the beginning. The condition expression is shown next to the sequence flow.

XML representation

A conditional sequence flow is represented in XML as a regular sequence flow, containing a conditionExpression sub-element. Note that for the moment only tFormalExpressions are supported. Omitting the xsi:type="" definition will simply default to this, the only supported type of expression.

```
1  <sequenceFlow id="flow" sourceRef="theStart" targetRef="theTask">
2    <conditionExpression xsi:type="tFormalExpression">
3      <![CDATA[${order.price > 100 && order.price < 250}]]>
4    </conditionExpression>
5  </sequenceFlow>
```

Currently conditionalExpressions can only be used with UEL; detailed info about these can be found in section Expressions. The expression used should resolve to a Boolean value, otherwise an exception is thrown while evaluating the condition.

- The example below references data of a process variable, in the typical JavaBean style through getters.

```
1  <conditionExpression xsi:type="tFormalExpression">
2    <![CDATA[${order.price > 100 && order.price < 250}]]>
3  </conditionExpression>
```

- This example invokes a method that resolves to a Boolean value.

```
1  <conditionExpression xsi:type="tFormalExpression">
2    <![CDATA[${order.isStandardOrder()}]]>
3  </conditionExpression>
```

5.3.5. Default Sequence Flow

Description

All BPMN 2.0 tasks and gateways can have a default sequence flow. A default sequence flow is only selected as the outgoing sequence flow for an activity if none of the other sequence flow could be selected. Conditions on a default sequence flow are always ignored.

Graphical notation

A default sequence flow is visualized as a regular sequence flow, with a slash marker at the beginning.

XML representation

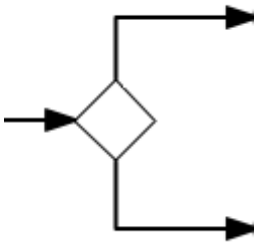
A default sequence flow for a certain activity is defined by the default attribute on that activity. The following XML snippet shows an example of an exclusive gateway that has the default sequence flow “flow 2”. Only when conditionA and conditionB both evaluate to false, will it be chosen as the outgoing sequence flow for the gateway.

```
4   <exclusiveGateway id="exclusiveGw" name="Exclusive Gateway" default="flow2" />
5   <sequenceFlow id="flow1" sourceRef="exclusiveGw" targetRef="task1">
6     <conditionExpression
    xsi:type="tFormalExpression">${conditionA}</conditionExpression>
7   </sequenceFlow>
8   <sequenceFlow id="flow2" sourceRef="exclusiveGw" targetRef="task2"/>
9   <sequenceFlow id="flow3" sourceRef="exclusiveGw" targetRef="task3">
10    <conditionExpression
    xsi:type="tFormalExpression">${conditionB}</conditionExpression>
11  </sequenceFlow>
```

5.4. *Gateways*

A gateway is used to control the flow of execution (or as the BPMN 2.0 specification describes, the tokens of execution). A gateway is capable of consuming or generating tokens.

A gateway is graphically visualized as a diamond shape with an icon inside. The icon shows the type of gateway.



5.4.1. Exclusive Gateway

Description

An exclusive gateway (also called the XOR gateway or more technically the exclusive data-based gateway) is used to model a decision in the process. When the execution arrives at this gateway, all outgoing sequence flow are evaluated in the order in which they are defined. The sequence flow with a condition evaluates to true (or which doesn’t have a condition set, conceptually having a 'true' defined on the sequence flow) is selected for continuing the process.

Note that the semantics of outgoing sequence flows is different to that of the general case in BPMN 2.0. While in general, all sequence flows with a condition that evaluates to true, are selected to continue in a parallel way. Only one sequence flow is selected when using an exclusive gateway. In the case of multiple sequence flows have a condition that evaluates to true, the first one defined in the XML (and only that one!) is selected for continuing the process. If no sequence flow can be selected, an exception will be thrown.

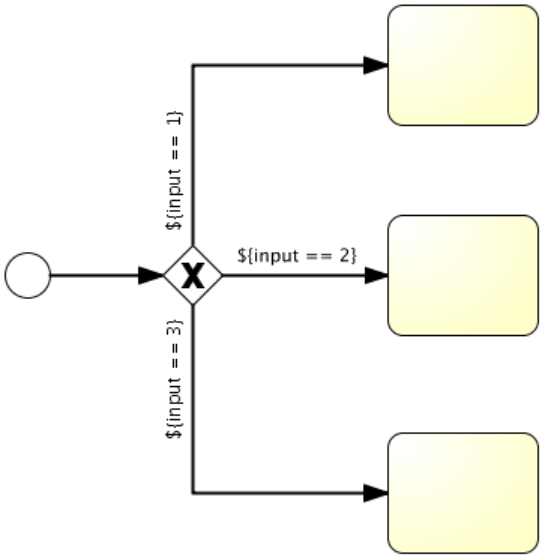
Graphical notation

An exclusive gateway is visualized as a typical gateway (i.e., a diamond shape) with an X icon inside, referring to the XOR semantics. Note that a gateway without an icon inside defaults to an exclusive gateway. The BPMN 2.0 specification does not allow mixing the diamond with and without an X in the same process definition.

XML representation

The XML representation of an exclusive gateway is straight-forward: one line defining the gateway and condition expressions defined on the outgoing sequence flows. See the section on conditional sequence flows to see which options are available for such expressions.

Take for example the following model:



Which is represented in XML as follows:

```
1  <exclusiveGateway id="exclusiveGw" name="Exclusive Gateway" />
2
3  <sequenceFlow id="flow2" sourceRef="exclusiveGw" targetRef="theTask1">
4    <conditionExpression xsi:type="tFormalExpression">${input ==
   1}</conditionExpression>
5  </sequenceFlow>
6
7  <sequenceFlow id="flow3" sourceRef="exclusiveGw" targetRef="theTask2">
8    <conditionExpression xsi:type="tFormalExpression">${input ==
   2}</conditionExpression>
9  </sequenceFlow>
10
11 <sequenceFlow id="flow4" sourceRef="exclusiveGw" targetRef="theTask3">
12   <conditionExpression xsi:type="tFormalExpression">${input ==
   3}</conditionExpression>
13 </sequenceFlow>
```

5.4.2. Parallel Gateway

Description

Gateways can also be used to model concurrency in a process. The most straightforward gateway to introduce concurrency in a process model is the Parallel Gateway. Parallel Gateways allow a process to fork into multiple paths of execution or join multiple incoming paths of execution.

The functionality of the parallel gateway is based on the incoming and outgoing sequence flows:

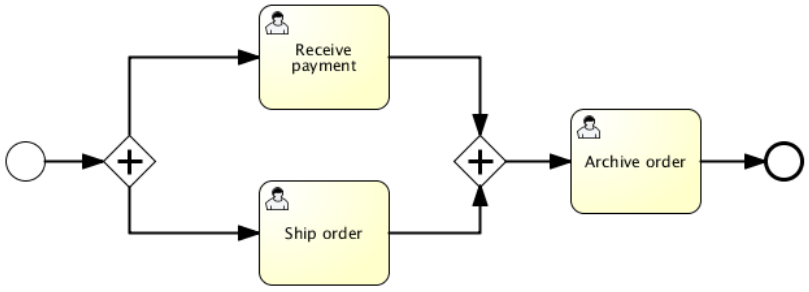
- **fork**: all outgoing sequence flows are followed in parallel, creating one concurrent execution for each sequence flow.
- **join**: all concurrent executions arriving at the parallel gateway wait in the gateway until an execution has arrived for each of the incoming sequence flow. After each incoming flow has arrived, the process then continues past the joining gateway.

Note that a parallel gateway can have both forking and joining behavior if there are multiple incoming and outgoing sequence flows for the same parallel gateway. In this case, the gateway will first join all incoming sequence flow before splitting into multiple concurrent paths of executions.

An important difference in contrast with other gateway types is that the parallel gateway does not evaluate conditions. If conditions are defined on the sequence flow connected with the parallel gateway, they are simply ignored.

Graphical Notation

A parallel gateway is visualized as a gateway (diamond shape) with the plus symbol inside referring to the AND semantics.



XML representation

Defining a parallel gateway needs one line of XML:

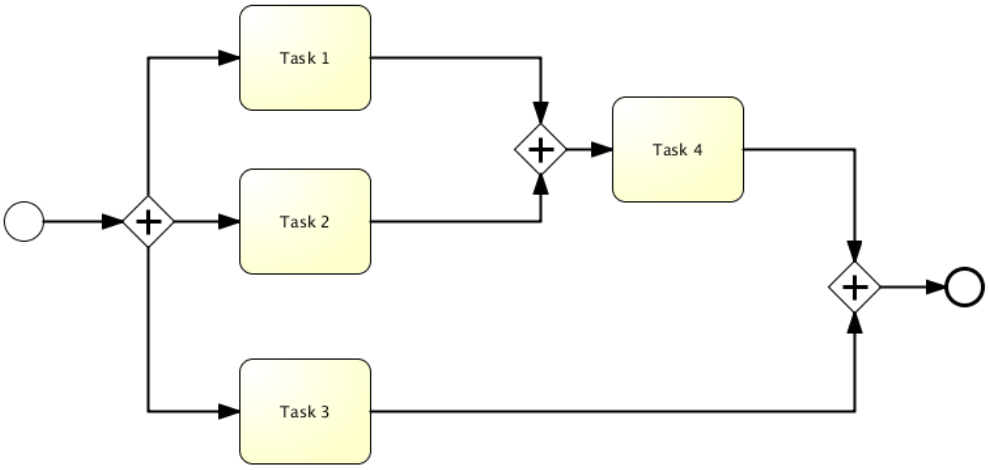
```
<parallelGateway id="myParallelGateway" />
```

The actual behavior (forking, joining or both), is defined by the sequence flow connected to the parallel gateway.

For example, the model above comes down to the following XML:

```
<startEvent id="theStart" />
<sequenceFlow id="flow1" sourceRef="theStart" targetRef="fork" />

<parallelGateway id="fork" />
<sequenceFlow sourceRef="fork" targetRef="receivePayment" />
<sequenceFlow sourceRef="fork" targetRef="shipOrder" />

<userTask id="receivePayment" name="Receive Payment" />
<sequenceFlow sourceRef="receivePayment" targetRef="join" />

<userTask id="shipOrder" name="Ship Order" />
<sequenceFlow sourceRef="shipOrder" targetRef="join" />

<parallelGateway id="join" />
<sequenceFlow sourceRef="join" targetRef="archiveOrder" />

<userTask id="archiveOrder" name="Archive Order" />
<sequenceFlow sourceRef="archiveOrder" targetRef="theEnd" />

<endEvent id="theEnd" />
```

Note that a parallel gateway does not need to be balanced (i.e., a matching number of incoming/outgoing sequence flow for corresponding parallel gateways). A parallel gateway will simply wait for all incoming sequence flows and create a concurrent path of execution for each outgoing sequence flow, not influenced by other constructs in the process model. So, the following process is legal in BPMN 2.0:



5.4.3. Inclusive Gateway

Description

The Inclusive Gateway can be seen as a combination of an exclusive and a parallel gateway. Like an exclusive gateway you can define conditions on outgoing sequence flows and the inclusive gateway will evaluate them. The main difference between the two is that the inclusive gateway can take more than one sequence flow, like the parallel gateway.

The functionality of the inclusive gateway is based on the incoming and outgoing sequence flows:

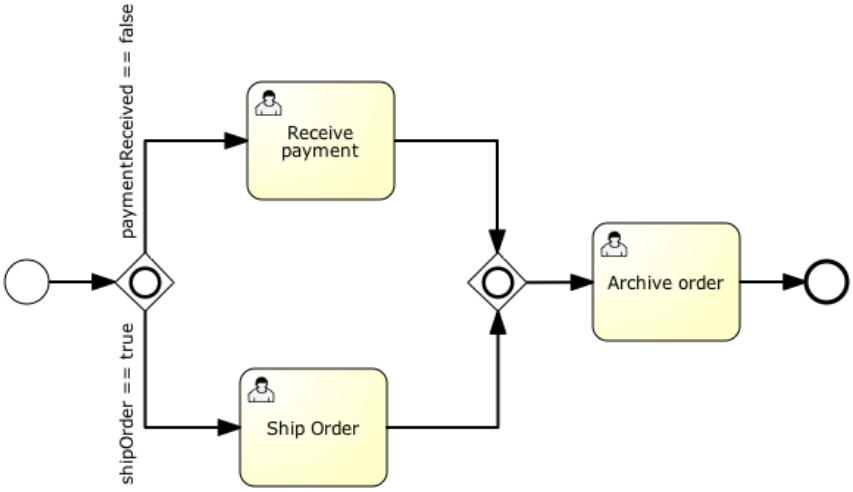
- **fork**: all outgoing sequence flow conditions are evaluated and for the sequence flow conditions that evaluate to true the flows are followed in parallel, creating one concurrent execution for each sequence flow.
- **join**: all concurrent executions arriving at the inclusive gateway wait in the gateway until an execution has arrived for each of the incoming sequence flows that have a process token. This is an important difference with the parallel gateway. So, in other words, the inclusive gateway will only wait for the incoming sequence flows that will be executed. After the join, the process continues past the joining inclusive gateway.

Note that an inclusive gateway can have both forking and joining behavior, if there are multiple incoming and outgoing sequence flows for the same inclusive gateway. In this case, the gateway will first join all incoming sequence flows that have a process token

before splitting into multiple concurrent paths of executions for the outgoing sequence flows that have a condition that evaluates to true.

Graphical Notation

An inclusive gateway is visualized as a gateway (diamond shape) with the circle symbol inside.



XML representation

Defining an inclusive gateway needs one line of XML:

```
1  <inclusiveGateway id="myInclusiveGateway" />
```

The actual behavior (forking, joining or both), is defined by the sequence flows connected to the inclusive gateway.

For example, the model above comes down to the following XML:

```
1  <startEvent id="theStart" />
2  <sequenceFlow id="flow1" sourceRef="theStart" targetRef="fork" />
3
4  <inclusiveGateway id="fork" />
5  <sequenceFlow sourceRef="fork" targetRef="receivePayment" >
6    <conditionExpression xsi:type="tFormalExpression">${paymentReceived ==
   false}</conditionExpression>
7  </sequenceFlow>
```

```
8   <sequenceFlow sourceRef="fork" targetRef="shipOrder" >
9     <conditionExpression xsi:type="tFormalExpression">${shipOrder ==
    true}</conditionExpression>
10  </sequenceFlow>
11
12  <userTask id="receivePayment" name="Receive Payment" />
13  <sequenceFlow sourceRef="receivePayment" targetRef="join" />
14
15  <userTask id="shipOrder" name="Ship Order" />
16  <sequenceFlow sourceRef="shipOrder" targetRef="join" />
17
18  <inclusiveGateway id="join" />
19  <sequenceFlow sourceRef="join" targetRef="archiveOrder" />
20
21  <userTask id="archiveOrder" name="Archive Order" />
22  <sequenceFlow sourceRef="archiveOrder" targetRef="theEnd" />
23
24  <endEvent id="theEnd" />
```

In the above example, after the process is started, two tasks will be created if the process variables paymentReceived == false and shipOrder == true. In case only one of these process variables equals to true, only one task will be created. If no condition evaluates to true an exception is thrown. This can be prevented by specifying a default outgoing sequence flow.

Note that an inclusive gateway does not need to be balanced (i.e., a matching number of incoming/outgoing sequence flows for corresponding inclusive gateways). An inclusive gateway will simply wait for all incoming sequence flows and create a concurrent path of execution for each outgoing sequence flow, not influenced by other constructs in the process model.

5.4.4. Event-based Gateway

Description

The Event-based Gateway allows for making decisions based on events. Each outgoing sequence flow of the gateway needs to be connected to an intermediate catching event. When process execution reaches an Event-based Gateway, the gateway acts like a wait state: execution is suspended. In addition, for each outgoing sequence flow, an event subscription is created.

Note the sequence flows running out of an Event-based Gateway are different from ordinary sequence flows. These sequence flows are never actually "executed". On the

contrary, they allow the process engine to determine which events an execution arriving at an Event-based Gateway needs to subscribe to. The following restrictions apply:

- An Event-based Gateway must have two or more outgoing sequence flows.
- An Event-based Gateway must only be connected to elements with the type intermediateCatchEvent. (Receive Tasks after an Event-based Gateway are not supported by TechDoc.)
- An intermediateCatchEvent connected to an Event-based Gateway must have a single incoming sequence flow.

Graphical notation

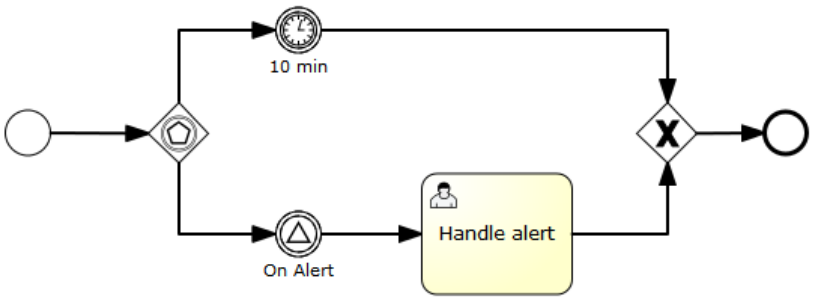
An Event-based Gateway is visualized as a diamond shape like other BPMN gateways with a special icon inside.

XML representation

The XML element used to define an Event-based Gateway is eventBasedGateway.

Example(s)

The following process is an example of a process with an Event-based Gateway. When the execution arrives at the Event-based Gateway, process execution is suspended. In addition, the process instance subscribes to the alert signal event and creates a timer which fires after 10 minutes. This effectively causes the process engine to wait for ten minutes for a signal event. If the signal occurs within 10 minutes, the timer is cancelled and execution continues after the signal. If the signal is not fired, execution continues after the timer and the signal subscription is cancelled.



```
<definitions id="definitions"
    xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
    targetNamespace="Examples">

    <signal id="alertSignal" name="alert" />

    <process id="catchSignal">

        <startEvent id="start" />

        <sequenceFlow sourceRef="start" targetRef="gw1" />

        <eventBasedGateway id="gw1" />

        <sequenceFlow sourceRef="gw1" targetRef="signalEvent" />
        <sequenceFlow sourceRef="gw1" targetRef="timerEvent" />

        <intermediateCatchEvent id="signalEvent" name="Alert">
            <signalEventDefinition signalRef="alertSignal" />
        </intermediateCatchEvent>

        <intermediateCatchEvent id="timerEvent" name="Alert">
            <timerEventDefinition>
                <timeDuration>PT10M</timeDuration>
            </timerEventDefinition>
        </intermediateCatchEvent>

        <sequenceFlow sourceRef="timerEvent" targetRef="exGw1" />
        <sequenceFlow sourceRef="signalEvent" targetRef="task" />

        <userTask id="task" name="Handle alert"/>

        <exclusiveGateway id="exGw1" />

        <sequenceFlow sourceRef="task" targetRef="exGw1" />
        <sequenceFlow sourceRef="exGw1" targetRef="end" />

        <endEvent id="end" />
</process>
</definitions>
```

5.5. *Tasks*

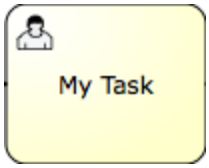
5.5.1. User Task

Description

A user task is used to model work that needs to be done by a human actor. When the process execution arrives at a user task, a new task is created in the task list of the user(s) or group(s) assigned to that task.

Graphical notation

A user task is visualized as a typical task (rounded rectangle), with a small user icon in the left upper corner.



XML representation

A user task is defined in XML as follows. The id attribute is required, the name attribute is optional.

```
1   <userTask id="theTask" name="Important task" />
```

A user task can have also a description. In fact, any BPMN 2.0 element can have a description. A description is defined by adding the documentation element.

```
1   <userTask id="theTask" name="Schedule meeting" >
2     <documentation>
3           Schedule an engineering meeting for next week with the new hire.
4     </documentation>
```

Due Date

Each task has a field, indicating the due date of that task. In TechDoc, there is an extension which allows you to specify an expression in your task-definition to set the initial due date of a task when it is created. The expression should always resolve to an ISO8601 time-duration (e.g., PT50M) or null. For example, you could use a date that was entered in a previous form in the process or calculated in a previous Service Task. If case a time-duration is used, the due-date is calculated based on the current time,

incremented by the given period. For example, when "PT30M" is used as the dueDate, the task is due in thirty minutes from now.

```
1  <userTask id="theTask" name="Important task"
   docubrain:dueDate="${dateVariable}"/>
```

User assignment

A user task can be directly assigned to a user. This is done by defining a humanPerformer sub element. A humanPerformer definition needs a resourceAssignmentExpression that actually defines the user. Currently, only formalExpressions are supported.

```
1  <process >
2
3    ...
4
5    <userTask id='theTask' name='important task' >
6      <humanPerformer>
7        <resourceAssignmentExpression>
8          <formalExpression>kermit</formalExpression>
9        </resourceAssignmentExpression>
10     </humanPerformer>
11   </userTask>
```

Only one user can be assigned as human performer to the task. In TechDoc terminology, this user is called the assignee and should be the user's username. Tasks that have an assignee are not visible in the task lists of other users or groups and may only be found in the task list of the specified user.

Tasks can also be put in the so-called candidate task list of people. In TechDoc terminology, this is referred to as the tasks available to claim. In this case, the potentialOwner construct must be used. The usage is similar to the humanPerformer construct. Do note that it is required to define for each element in the formal expression to specify if it is a user or a group (the engine cannot guess this).

```
1  <process >
2
3    ...
4
5    <userTask id='theTask' name='important task' >
6      <potentialOwner>
7        <resourceAssignmentExpression>
```

```
8          <formalExpression>user(kermit), group(management)</formalExpression>
9        </resourceAssignmentExpression>
10     </potentialOwner>
11   </userTask>
```

This will retrieve all tasks where kermit is a candidate user, i.e., the formal expression contains user(kermit). This will also retrieve all tasks that are assigned to a group where kermit is a member of (e.g., group(management), if kermit is a member of that group.

If no specifics are given whether the given text string is a user or group, the engine defaults to group. So, the following would be the same as when group(accountancy) was declared.

```
1   <formalExpression>accountancy</formalExpression>
```

TechDoc extensions for task assignment

It is clear that user and group assignments are quite cumbersome for use cases where the assignment is not complex. To avoid these complexities, custom extensions on the user task are possible.

- **assignee attribute**: this custom extension allows to directly assign a user task to a given user.

```
1   <userTask id="theTask" name="my task" docubrain:assignee="kermit" />
```

This is exactly the same as using a humanPerformer construct as defined above.

- **candidateUsers attribute**: this custom extension allows to make a user a candidate for a task.

```
1   <userTask id="theTask" name="my task" docbrain:candidateUsers="kermit, gonzo"
    />
```

This is exactly the same as using a potentialOwner construct as defined above. Note that it is not required to use the user(kermit) declaration as is the case with the potential owner construct, since the attribute can only be used for users.

- **candidateGroups attribute**: this custom extension allows to make a group a candidate for a task.

```
1   <userTask id="theTask" name="my task"
    docubrain:candidateGroups="management, accountancy" />
```

This is exactly the same as using a potentialOwner construct as defined above. Note that it is not required to use the group(management) declaration as is the case with the potential owner construct, since the attribute can only be used for groups.

- candidateUsers and candidateGroups can both be defined on the same user task.

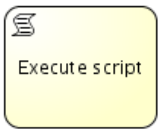
5.5.2. Script Task

Description

A script task is an automatic activity. When a process execution arrives at the script task, the corresponding script is executed. TechDoc only supports using JavaScript within a script task. Along with all of the fundamentals of JavaScript, TechDoc also includes a few helper objects that add additional functionalities. These helper objects can provide things like easy-to-use base 64 encoding, HTML parsing, JSON parsing/construction, etc. See the Script Task Helper Objects section for more information on these.

Graphical Notation

A script task is visualized as a typical BPMN 2.0 task (rounded rectangle), with a small script icon in the top-left corner of the rectangle.



XML representation

A script task is defined by specifying the script and the scriptFormat.

Variables in scripts

All process variables that are accessible through the execution that arrives in the script task, can be used within the script. In the example, the script variable 'inputArray' is in fact a process variable (an array of integers).

```
1  <scriptTask id="theScriptTask" name="Execute script" scriptFormat="JavaScript">
2    <script>
3      var sum = 0;
```

```
4      for ( i in inputArray )
5      {
6        sum += I;
7      }
8     </script>
9   </scriptTask>
```

It's also possible to set process variables in a script, simply by calling execution.setVariable("variableName", variableValue). By default, no variables are stored automatically.

Example on how to set a variable in a script:

```
1   <script>
2      var scriptVar = "test123";
3      execution.setVariable("myVar", scriptVar);
4   </script>
```

JUEL Expressions

Inside of JavaScript based script tasks, JUEL (Java Unified Expression Language) expressions can also be used. JUEL is used as a variable substitution method to work with dynamic values. For example:

```
1   <script>
2      var scriptVar = "Working on document ";
3      scriptVar += ${document};
4      execution.setVariable("myVar", scriptVar);
5   </script>
```

Similar to the previous code example where we set a JavaScript value into a process variable, in this example we've gone a step farther and appended information about the document the current process instance is working on by specifying the document bean from the process variable named document. Before a script task (or any type of task) is executed, JUEL is always called first to evaluate any JUEL expressions. JUEL expressions are specified using the standard convention ${}. Once these expressions have been evaluated, the resulting value is substituted in place. So, the previous example becomes:

```
1   <script>
2      var scriptVar = "Working on document ";
3      scriptVar += “123”;
4      execution.setVariable("myVar", scriptVar);
5   </script>
```

After JUEL has processed the entire script block, the JavaScript then executes the JavaScript code. Alternatively, similar to execution.setVariable, execution.getVariable may be used as well:

```
1  <script>
2     var scriptVar = "Working on document ";
3     scriptVar += execution.getVariable("document");
4  </script>
```

JUEL has other functions, but it's main use is to provide these types of simplistic substitutions. For more information on JUEL, please review the Java Unified Expression Language specification.

Script results

The return value of a script task can be assigned to an already existing or to a new process variable by specifying the process variable name as a literal value for the 'docubrain:resultVariable' attribute of a script task definition. Any existing value for a specific process variable will be overwritten by the result value of the script execution. When not specifying a result variable name, the script result value gets ignored.

```
1  <scriptTask id="theScriptTask" name="Execute script" scriptFormat="JavaScript"
   docubrain:resultVariable="myVar">
2    <script>var testVar = “test”; return testVar;</script>
3  </scriptTask>
```

In the above example, the result of the script execution (the returned value of testVar in this case) is set to the process variable named 'myVar' after the script completes.

5.5.3. Service Task

Description

A service task is used to invoke TechDoc-specific service task operations, call custom built Java classes included in the TechDoc Java environment, perform basic tasks via expressions, and call SOAP based web services.

Graphical Notation

A service task is visualized as a rounded rectangle with a small gear icon in the top-left corner.

XML representation

There are 4 ways of declaring different usages of a service task:

- Specifying a class that implements JavaDelegate or ActivityBehavior
- Evaluating an expression that resolves to a delegation object
- Invoking a method expression or evaluating a value expression
- Calling web services

Java Delegate

To specify a class that is called during process execution, the fully qualified classname needs to be provided by the docubrain:class attribute.

```
1  <serviceTask id="javaService" name="My Java Service Task"
   docubrain:class="wm.activity.ReserveDocument" >
2        <docubrain:field name="DocNumber">
3          <docubrain:expression><![CDATA[123]]></docubrain:expression>
4        </docubrain:field>
5        <docubrain:field name="GenNumber">
6          <docubrain:expression><![CDATA[1.0]]></docubrain:expression>
7        </docubrain:field>
8        <docubrain:field name="Reason">
9          <docubrain:expression><![CDATA[Reserved document via
   workflow.]]></docubrain:expression>
10       </docubrain:field>
11     </extensionElements>
12 </serviceTask>
```

In the example above, a TechDoc activity is being called to reserve a document with the document number “123”. See the TechDoc Service Task Operations section for more information on all of the TechDoc operations that can be performed from a service task. As mentioned above, custom classes can be developed and used in the same fashion to perform customer-specific tasks. For more information on custom development, please

contact us using the information at the bottom of this document or by visiting docubrain.com.

Delegate Expression

Delegate expressions are very similar to referencing Java delegates as we did in the previous example. In the previous example, we referenced the Java class to call by specifying the full name of the class. However, it is also possible to call Java delegates that have been injected into a process as a variable by using the syntax:

```
1   <serviceTask id="serviceTask"
    docubrain:delegateExpression="${delegateExpressionBean}" />
```

The ${} syntax being used here is JUEL (Java Unified Expression Language). In this example, instead of referencing the class directly by name, we are referencing an instance of the class that has been injected into the process as an object using the process variable name “delegateExpressionBean”. Currently, all of the available TechDoc Service Task operations are only available using the docubrain:class attribute as stated in the previous example. However, as mentioned previously, custom code can be developed and included so that it may be referenced in this manner. For instance, if a Java delegate needed custom initialization that was specific to each process instance, this initialization could be performed prior to a process starting and then injected in the process as a variable for later use by a delegate expression.

Typically, delegate expressions are only used when needed (as mentioned in the example of needing process specific initialization) because they introduce additional process variables, use additional process storage (to store the object itself), etc.

Expressions

Method expressions can also be used within a service task and the usage is very similar to that of Script Task:

```
1   <serviceTask id="javaService"
2           name="My Java Service Task"
3           docubrain:expression="#{printer.printMessage()}" />
```

Method printMessage (without parameters) will be called on the named object called printer. However, instead of using the typical JUEL syntax, a simpler syntax is used #{} called UEL (Unified Expression Language). This syntax has less capabilities than the full JUEL engine and is setup specifically for calling a single method on an already existing object stored in a variable on a process. In the example above, a Java delegate instance stored under the process variable printer is being referenced and its printMessage

method is being called. Since TechDoc does not make heavy use of delegate expressions, method expressions are not commonly used either.

It's also possible to pass parameters with a method used in the expression.

```
<serviceTask id="javaService"
        name="My Java Service Task"
        docubrain:expression="#{printer.printMessage(execution, myVar)}" />
```

Method printMessage will be called on the object named printer. The first parameter passed is the DelegateExecution, which is available in the expression context by default available as execution. The second parameter passed, is the value of the variable with name myVar in the current execution.

To specify a UEL value expression that should be evaluated, use attribute docubrain:expression.

```
<serviceTask id="javaService"
        name="My Java Service Task"
        docubrain:expression="#{split.ready}" />
```

The getter method of property ready, will be called on the named bean called split. The named objects are resolved in the execution's process variables.

Web Services

Web services may also be called via service tasks. To use a web service, we need to import its operations and complex types. This can be done automatically by using the import tag pointing to the WSDL of the Web service:

```
<import importType="http://schemas.xmlsoap.org/wsdl/"
    location="http://localhost:1234/counter?wsdl"
    namespace="http://webservice.example.org/" />
```

The previous declaration tells TechDoc to import the definitions but it doesn't create the item definitions and messages for you. Let's suppose we want to invoke a specific method called prettyPrint, therefore we will need to create the corresponding message and item definitions for the request and response messages:

```
<message id="prettyPrintCountRequestMessage"
itemRef="tns:prettyPrintCountRequestItem" />
<message id="prettyPrintCountResponseMessage"
itemRef="tns:prettyPrintCountResponseItem" />
```

```
6
7   <itemDefinition id="prettyPrintCountRequestItem"
    structureRef="counter:prettyPrintCount" />
8   <itemDefinition id="prettyPrintCountResponseItem"
    structureRef="counter:prettyPrintCountResponse" />
```

Before declaring the service task, we have to define the BPMN interfaces and operations that actually reference the Web service ones. For each operation we reuse the previous defined message for in and out. For example, the following declaration defines the counter interface and the prettyPrintCountOperation operation:

```
1   <interface name="Counter Interface" implementationRef="counter:Counter">
2       <operation id="prettyPrintCountOperation" name="prettyPrintCount Operation"
3                       implementationRef="counter:prettyPrintCount">
4       <inMessageRef>tns:prettyPrintCountRequestMessage</inMessageRef>
5       <outMessageRef>tns:prettyPrintCountResponseMessage</outMessageRef>
6       </operation>
7   </interface>
```

Then we can declare a Web Service Task by using the ##WebService implementation and a reference to the Web service operation.

```
1   <serviceTask id="webService"
2       name="Web service invocation"
3       implementation="##WebService"
4       operationRef="tns:prettyPrintCountOperation">
```

Web Service Task IO Specification

Unless we are using the simplistic approach for data input and output associations (See below), each Web Service Task needs to declare an IO Specification which states which are the inputs and outputs of the task. The approach is pretty straightforward and BPMN 2.0 complaint, for our prettyPrint example we define the input and output sets according to the previously declared item definitions:

```
1   <ioSpecification>
2       <dataInput itemSubjectRef="tns:prettyPrintCountRequestItem"
    id="dataInputOfServiceTask" />
3       <dataOutput itemSubjectRef="tns:prettyPrintCountResponseItem"
    id="dataOutputOfServiceTask" />
4       <inputSet>
5               <dataInputRefs>dataInputOfServiceTask</dataInputRefs>
6       </inputSet>
```

```
7       <outputSet>
8               <dataOutputRefs>dataOutputOfServiceTask</dataOutputRefs>
9       </outputSet>
10  </ioSpecification>
```

Web Service Task data input associations

There are 2 ways of specifying data input associations:

- Using expressions
- Using the simplistic approach

To specify the data input association using expressions we need to define the source and target items and specify the corresponding assignments between the fields of each item. In the following example we assign prefix and suffix fields of the items:

```
1   <dataInputAssociation>
2       <sourceRef>dataInputOfProcess</sourceRef>
3       <targetRef>dataInputOfServiceTask</targetRef>
4       <assignment>
5               <from>${dataInputOfProcess.prefix}</from>
6               <to>${dataInputOfServiceTask.prefix}</to>
7       </assignment>
8       <assignment>
9               <from>${dataInputOfProcess.suffix}</from>
10              <to>${dataInputOfServiceTask.suffix}</to>
11      </assignment>
12  </dataInputAssociation>
```

On the other hand, we can use the simplistic approach which is much simpler. The sourceRef element is a TechDoc variable name and the targetRef element is a property of the item definition. In the following example we assign to the prefix field the value of the variable PrefixVariable and to the suffix field the value of the variable SuffixVariable.

```
1   <dataInputAssociation>
2       <sourceRef>PrefixVariable</sourceRef>
3       <targetRef>prefix</targetRef>
4   </dataInputAssociation>
5   <dataInputAssociation>
6       <sourceRef>SuffixVariable</sourceRef>
7       <targetRef>suffix</targetRef>
8   </dataInputAssociation>
```

Web Service Task data output associations

There are 2 ways of specifying data out associations:

- Using expressions
- Using the simplistic approach

To specify the data out association using expressions we need to define the target variable and the source expression. The approach is pretty straightforward and similar data input associations:

```
1  <dataOutputAssociation>
2      <targetRef>dataOutputOfProcess</targetRef>
3      <transformation>${dataOutputOfServiceTask.prettyPrint}</transformation>
4  </dataOutputAssociation>
```

On the other hand, we can use the simplistic approach which is much simpler. The sourceRef element is a property of the item definition and the targetRef element is a TechDoc variable name. The approach is pretty straightforward and similar data input associations:

```
1  <dataOutputAssociation>
2      <sourceRef>prettyPrint</sourceRef>
3      <targetRef>OutputVariable</targetRef>
4  </dataOutputAssociation>
```

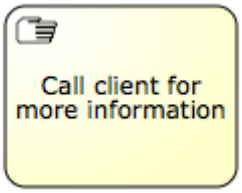
5.5.4. Manual Task

Description

A manual task defines a task that is external to the BPM engine. It is used to model work that is done by somebody, which the engine does not need to know of, nor is there a system or UI interface. For the engine, a manual task is handled as a pass-through activity, automatically continuing the process from the moment process execution arrives into it.

Graphical Notation

A manual task is visualized as a rounded rectangle, with a little hand icon in the upper left corner



XML representation

```
1    <manualTask id="myManualTask" name="Call client for more information" />
```

5.5.5. Receive Task

Description

A receive task is a simple task that waits for the arrival of a certain message. When process execution arrives at a Receive Task, the process state is committed to the persistence store. This means that the process will stay in this wait state, until a specific message is received by the engine, which triggers the continuation of the process past the Receive Task.

Graphical notation

A receive task is visualized as a task (rounded rectangle) with a message icon in the top left corner. The message is white (a black message icon would have send semantics)

XML representation

```
1    <receiveTask id="waitState" name="wait" />
```

5.5.6. Send Task

Description

A send task is a simple task that broadcasts a message in the workflow engine. This message can be received by other Receive Tasks in the same process or any other processes in the process engine.

Graphical notation

A send task is visualized as a task (rounded rectangle) with a message icon in the top left corner. The message is black (a white message icon would have receive semantics)

XML representation

```
1   <sendTask id="sendMsg" name="send" />
```

5.5.7. Shell Task

Description

The shell task allows to run shell scripts and commands. Note that the Shell task is not an official task of BPMN 2.0 spec (and it does not have a dedicated icon as a consequence).

Defining a shell task

The shell task is implemented as a dedicated Service Task and is defined by setting 'shell' for the type of the service task.

```
1   <serviceTask id="shellEcho" docubrain:type="shell">
```

The shell task is configured by field injection. All the values for these properties can contain EL expressions, which are resolved at runtime during process execution. The following properties can be set:

Property	Required?	Type	Description	Default
command	Yes	String	Shell command to execute	
arg0-5	No	String	Parameter 0 to Parameter 5	
wait	No	Boolean	True if the workflow	true

			process should wait until the shell process has terminated, false otherwise	
redirectError	No	Boolean	True if standard error should be merged with standard output, false otherwise	false
cleanEnv	No	Boolean	True if the shell process should inherit a clean environment, false otherwise	false
outputVariable	No	String	Name of the workflow process variable to receive the output of the shell process	Output is not recorded
errorCodeVariable	No	String	Name of the workflow process variable to receive the result error code of the shell process	Error code is not recorded
directory	No	String	Sets the working directory the shell process should execute under	Current directory

Example usage

The following XML snippet shows an example of using the shell Task. It runs shell script "cmd /c echo EchoTest", waits for it to be terminated and puts the result in resultVar:

```
1   <serviceTask id="shellEcho" docubrain:type="shell" >
2     <extensionElements>
3       <docubrain:field name="command" stringValue="cmd" />
4       <docubrain:field name="arg1" stringValue="/c" />
5       <docubrain:field name="arg2" stringValue="echo" />
6       <docubrain:field name="arg3" stringValue="EchoTest" />
7       <docubrain:field name="wait" stringValue="true" />
8       <docubrain:field name="outputVariable" stringValue="resultVar" />
9     </extensionElements>
10  </serviceTask>
```

5.6. *Sub-Processes and Call Activities*

5.6.1. Sub-Process

Description

A Sub-Process is an activity that contains other activities, gateways, events, etc. which on itself form a process that is part of the bigger process. A Sub-Process is completely defined inside a parent process (that's why it's often called an embedded Sub-Process).

Sub-Processes have two major use cases:

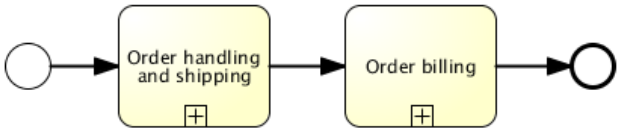
- Sub-Processes allow hierarchical modeling.
- A Sub-Process creates a new scope for events. Events that are thrown during execution of the Sub-Process, can be caught by a boundary event on the boundary of the Sub-Process, thus creating a scope for that event limited to the Sub-Process.

Using a Sub-Process does impose some constraints:

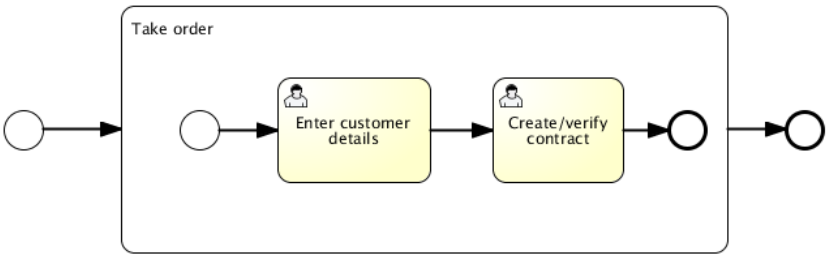
- A Sub-Process can only have one none start event; no other start event types are allowed. A Sub-Process must at least have one end event. Note that the BPMN 2.0 specification allows to omit the start and end events in a Sub-Process, but TechDoc does not support this.
- Sequence flow cannot cross Sub-Process boundaries.

Graphical Notation

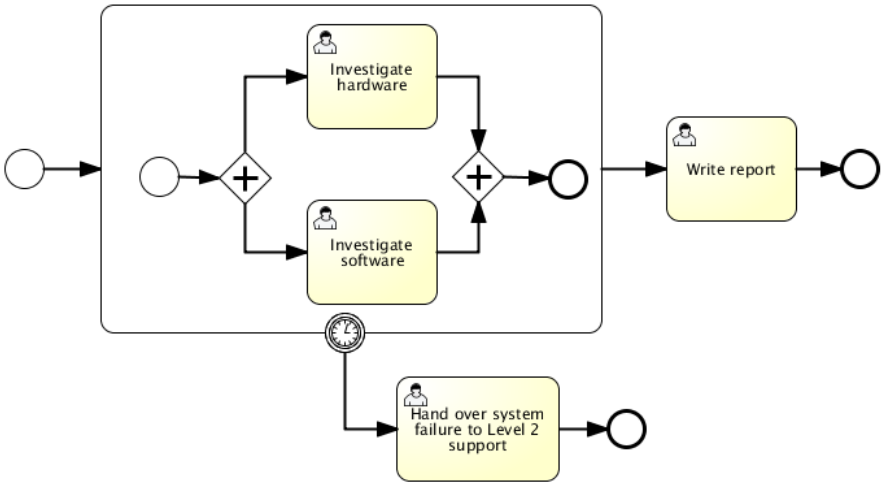
A Sub-Process is visualized as a typical activity, i.e., a rounded rectangle. In case the Sub-Process is collapsed, only the name and a plus-sign are displayed, giving a high-level overview of the process:



In case the Sub-Process is expanded, the steps of the Sub-Process are displayed within the Sub-Process boundaries:



One of the main reasons to use a Sub-Process, is to define a scope for a certain event. The following process model shows this: both the investigate software/investigate hardware tasks need to be done in parallel, but both tasks need to be done within a certain time, before Level 2 support is consulted. Here, the scope of the timer (i.e., which activities must be done in time) is constrained by the Sub-Process.



XML representation

A Sub-Process is defined by the subprocess element. All activities, gateways, events, etc. that are part of the Sub-Process, need to be enclosed within this element.

```
1  <subProcess id="subProcess">
2    <startEvent id="subProcessStart" />
3      ... other Sub-Process elements ...
4    <endEvent id="subProcessEnd" />
5  </subProcess>
```

5.6.2. Event Sub-Process

Description

The Event Sub-Process is new in BPMN 2.0. An Event Sub-Process is a subprocess that is triggered by an event. An Event Sub-Process can be added at the process level or at any subprocess level. The event used to trigger an event subprocess is configured using a start event. From this, it follows that none start events are not supported for Event Sub-Processes. An Event Sub-Process might be triggered using events like message events, error events, signal events, timer events, or compensation events. The subscription to the start event is created when the scope (process instance or subprocess) hosting the Event Sub-Process is created. The subscription is removed when the scope is destroyed.

An Event Sub-Process may be interrupting or non-interrupting. An interrupting subprocess cancels any executions in the current scope. A non-interrupting Event Sub-Process spawns a new concurrent execution. While an interrupting Event Sub-Process can only be triggered once for each activation of the scope hosting it, a non-interrupting

Event Sub-Process can be triggered multiple times. The fact whether the subprocess is interrupting is configured using the start event triggering the Event Sub-Process.

An Event Sub-Process must not have any incoming or outgoing sequence flows. Since an Event Sub-Process is triggered by an event, an incoming sequence flow makes no sense. When an Event Sub-Process is ended, either the current scope is ended (in case of an interrupting Event Sub-Process), or the concurrent execution spawned for the non-interrupting subprocess is ended.

Current limitations:

- TechDoc only supports interrupting Event Sub-Processes.
- TechDoc only supports Event Sub-Process triggered using an Error Start Event or Message Start Event.

Graphical Notation

An Event Sub-Process might be visualized as a an embedded subprocess with a dotted outline.

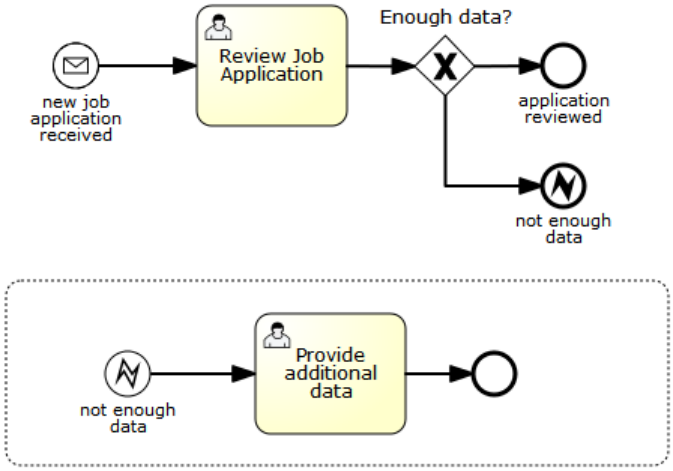
XML representation

An Event Sub-Process is represented using XML in the same way as a an embedded subprocess. In addition, the attribute triggeredByEvent must have the value true:

```
1   <subProcess id="eventSubProcess" triggeredByEvent="true">
2       ...
3   </subProcess>
```

Example

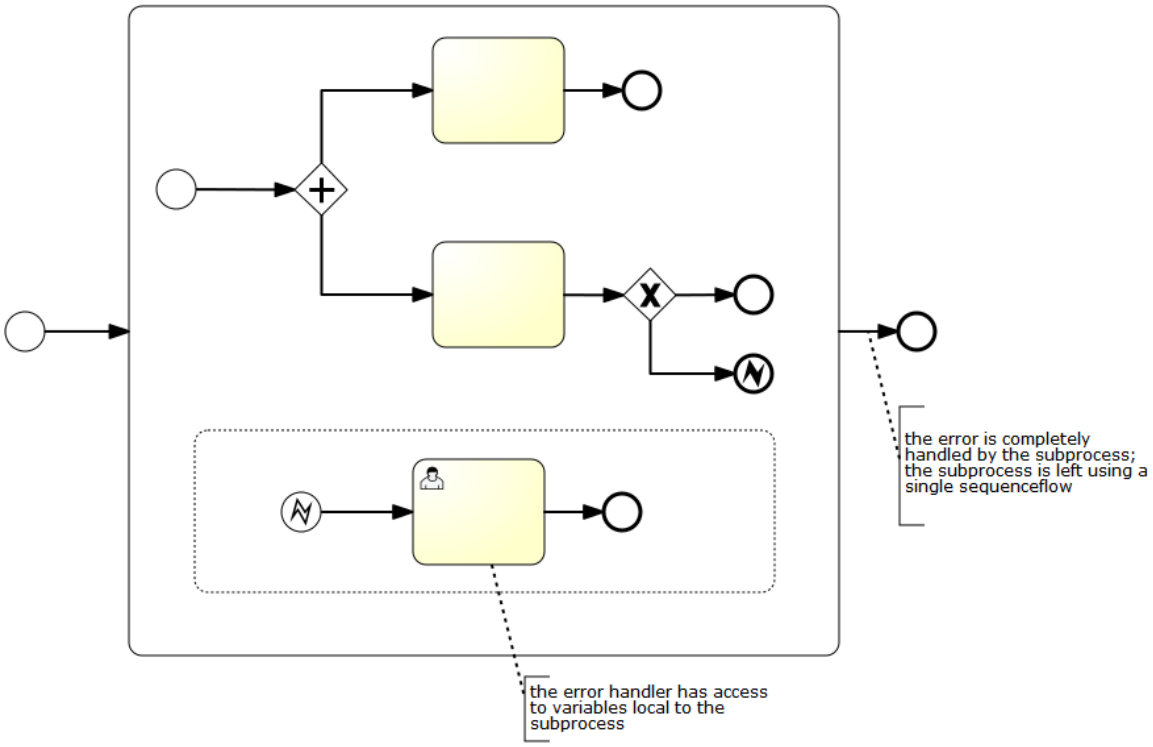
The following is an example of an Event Sub-Process triggered using an Error Start Event. The Event Sub-Process is located at the "process level", i.e., is scoped to the process instance:



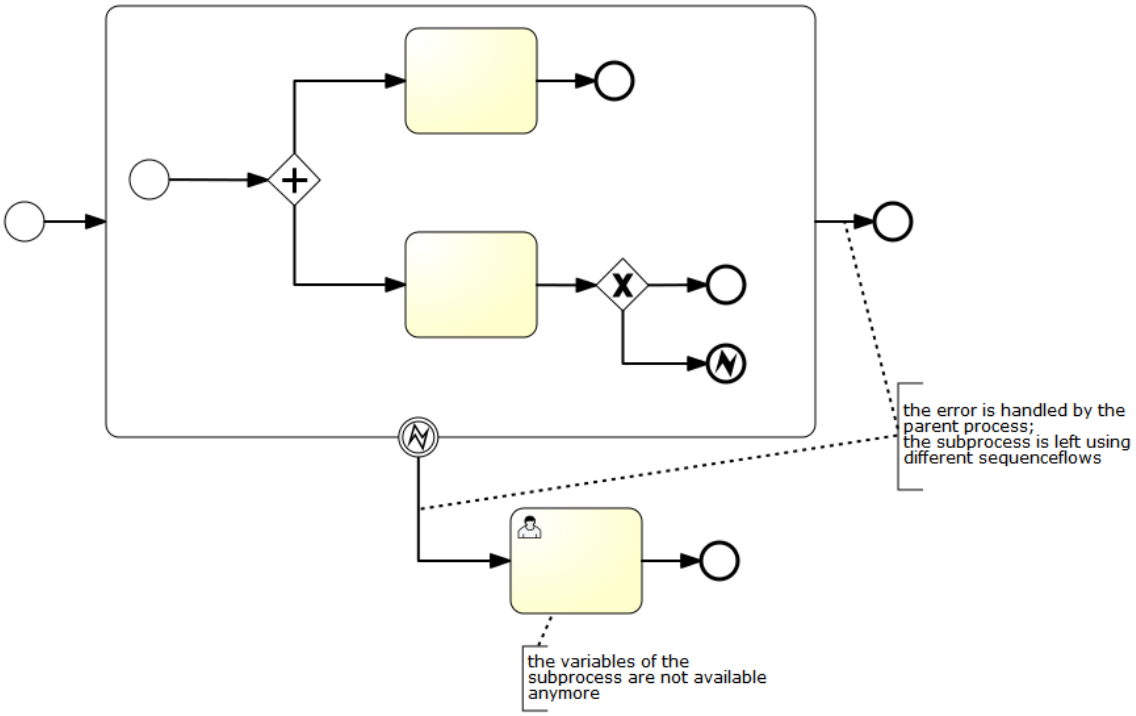
This is how the Event Sub-Process would look like in XML:

```
1  <subProcess id="eventSubProcess" triggeredByEvent="true">
2      <startEvent id="catchError">
3              <errorEventDefinition errorRef="error" />
4      </startEvent>
5      <sequenceFlow id="flow2" sourceRef="catchError"
   targetRef="taskAfterErrorCatch" />
6      <userTask id="taskAfterErrorCatch" name="Provide additional data" />
7  </subProcess>
```

As already stated, an Event Sub-Process can also be added to an embedded subprocess. If it is added to an embedded subprocess, it becomes an alternative to a boundary event. Consider the two following process diagrams. In both cases the embedded subprocess throws an error event. Both times the error is caught and handled using a user task.



as opposed to:



In both cases the same tasks are executed. However, there are differences between both modelling alternatives:

- The embedded subprocess is executed using the same execution of the scope it is hosted in. This means that an embedded subprocess has access to the variables local of its scope. When using a boundary event, the execution created for executing the embedded subprocess is deleted by the sequence flow leaving the boundary event. This means that the variables created by the embedded subprocess are not available anymore.

- When using an Event Sub-Process, the event is completely handled by the subprocess it is added to. When using a boundary event, the event is handled by the parent process.

These two differences can help you decide whether a boundary event or an embedded subprocess is better suited for solving a particular process modeling / implementation problem.

5.6.3. Call activity (subprocess)

Description

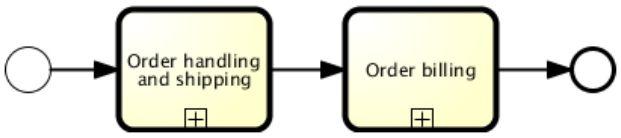
BPMN 2.0 makes a distinction between a regular subprocess, often also called embedded subprocess, and the call activity, which looks very similar. From a conceptual point of view, both will call a subprocess when process execution arrives at the activity.

The difference is that the call activity references a process that is external to the process definition, whereas the subprocess is embedded within the original process definition. The main use case for the call activity is to have a reusable process definition that can be called from multiple other process definitions.

When process execution arrives in the call activity, a new execution is created that is a sub-execution of the execution that arrives in the call activity. This sub-execution is then used to execute the subprocess, potentially creating a parallel child execution within the parent process. The super-execution waits until the subprocess has completely ended before continuing the parent process.

Graphical Notation

A call activity is visualized the same as a subprocess, however with a thick border (collapsed and expanded).



XML representation

A call activity is a regular activity, that requires a calledElement that references a process definition by its key. In practice, this means that the id of the process is used in the calledElement.

```
1  <callActivity id="callCheckCreditProcess" name="Check credit"
   calledElement="checkCreditProcess" />
```

Note that the process definition of the subprocess is resolved at runtime. This means that the subprocess can be deployed independently from the calling process, if needed.

Passing variables

You can pass process variables to the sub process and vice versa. The data is copied into the subprocess when it is started and copied back into the main process when it ends.

```
1  <callActivity id="callSubProcess" calledElement="checkCreditProcess" >
2    <extensionElements>
3        <docubrain:in source="someVariableInMainProcess"
   target="nameOfVariableInSubProcess" />
4        <docubrain:out source="someVariableInSubProcess"
   target="nameOfVariableInMainProcess" />
5    </extensionElements>
6  </callActivity>
```

We use a TechDoc Extension as a shortcut for the BPMN standard elements called dataInputAssociation and dataOutputAssociation, which only work if you declare process variables in the BPMN 2.0 standard way.

It is possible to use expressions here as well:

```
1  <callActivity id="callSubProcess" calledElement="checkCreditProcess" >
2      <extensionElements>
3        <docubrain:in sourceExpression="${x+5}" target="y" />
4        <docubrain:out source="${y+5}" target="z" />
5      </extensionElements>
```

```
6   </callActivity>
```

So, in the end z = y+5 = x+5+5

6. TechDoc Service Task Operations

In this section, we will discuss the TechDoc operations that are available for use from a Service Task within a workflow Process. If you are not familiar with using Service Tasks to call TechDoc Activities, please first read the previous section.

6.1. *Add Access Association*

Add Access Association allows you to add access associations to a Document for multiple Groups, Users, and Remote Users in a single call.

Service Class: *wm.activity.AddAccessAssociation*

Fields:

- *Groups* – (Optional) – The Groups field is used to specify access to be added for one or more Groups separated by semicolons. The value of this field should be specified as:

  groupName:accessSetting, accessSetting;

  For example, to associate a Group named *TestGroup* with Read and Modify access, you would specify:

  TestGroup:Read,Modify;

  If you wanted to add an additional Group named *AnotherGroup* to the previous example with Owner access you would specify:

  TestGroup:Read,Modify;AnotherGroup:Owner;

- *Users* – (Optional) – The *Users* field is used to specify access to be added for one or more Users separated by semicolons. The value of this field should be specified just like the Groups field:

  username:accessSetting, accessSetting;

  For Example, to associate a User named Joe with Delete access, you would specify:

  Joe:Delete;

If you wanted to add an additional User named Bob to the previous example with Reserve/Replace and Delete access you would specify:

Joe:Delete;Bob:Reserve/Replace,Delete;

- *RemoteUsers* – (Optional) – The *RemoteUsers* field is used to specify access to be added for one or more Remote Users separated by semicolons. Remote Users do not have access values when associating. Any associated Remote user is given Read access only. The value of this field should be specified as:

  *authenticatorName**username;*

  For example, to associate a Remote User named Joe that authenticates using the Authenticator named *TestAuthenticator* you would specify:

  *TestAuthenticator**Joe;*

  Remote Users can also be specified as Remote User name @ Authenticator name. For example:

  joe@TestAuthenticator;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

  At the time of the writing of this guide, the valid Document Access values are

  - None – A User has no access to the Document.
  - Read – A User can view the attributes of this Document, view the attributes of this Document's Generations and Renditions and fetch this Document's Generations and Renditions.
  - Modify – A User can modify the attributes of this Document and this Document's Generations and Renditions. The User must also have the Modify Document privilege.
  - Delete – A User can delete this Document. The User must also have the Delete Document privilege.
  - Reserve/Replace – A User can reserve and replace the Document. The User must also have the Create Generation privilege.

- Owner – A User can act as the owner of this Document. The User will be able to do anything to the Document that the owner can, as long as the User's privileges allow it.

6.2. *Add Commenter Association*

Add Commenter Association allows you to add commenter associations to a Document for multiple Groups and Users in a single call.

Service Class: *wm.activity.AddCommenterAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify one or more Groups to be added separated by semicolons. The value of this field should be specified as:

  groupName;groupName;

  For example, to associate a Group named TestGroup you would specify:

  TestGroup;

  If you wanted to add an additional Group named AnotherGroup to the previous example, you would specify:

  TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify one or more Users to be added separated by semicolons. The value of this field should be specified just like the Groups field:

  username;username;

  For Example, to associate a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User named Bob to the previous example, you would specify:

  Joe;Bob;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.3. *Add Distribution Association*

Add Distribution Association allows you to add distribution associations to a Document for multiple Groups, Users, and Remote Emails in a single call.

Service Class: *wm.activity.AddDistributionAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify one or more Groups to be added separated by semicolons. The value of this field should be specified as:

  groupName;groupName;

  For example, to associate a Group named TestGroup you would specify:

  TestGroup;

  If you wanted to add an additional Group named AnotherGroup to the previous example, you would specify:

  TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify one or more Users to be added separated by semicolons. The value of this field should be specified just like the Groups field:

  username;username;

  For Example, to associate a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User named Bob to the previous example, you would specify:

  Joe;Bob;

- *RemoteEmails* – (Optional) – The *RemoteEmails* field is used to specify one or more Remote Emails to be added separated by semicolons. The value of this field should be specified just like Groups and Users:

  remoteEmail;remoteEmail;

  For example, to associate the Remote Email joe@somewhere.com you would specify:

  joe@somewhere.com;

  If you wanted to add an additional Remote bob@somewhere.com to the previous example, you would specify:

  joe@somewhere.com;bob@somewhere.com;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.4. *Add Keyword*

Add Keyword allows you to add a Keyword to a Document.

Service Class: *wm.activity.AddKeyword*

Fields:

- *KeywordName* – (Mandatory) – The *KeywordName* field is used to specify the name of the Keyword to add.

- *KeywordValue* – (Mandatory) – The *KeywordValue* field is used to specify the value this Keyword should have.

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.5. *Add Notification Association*

Add Notification Association allows you to add notification associations to a Document for multiple Groups, Users, and Remote Emails in a single call.

Service Class: *wm.activity.AddNotificationAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify one or more Groups to be added separated by semicolons. The value of this field should be specified as:

  groupName;groupName;

  For example, to associate a Group named TestGroup you would specify:

  TestGroup;

  If you wanted to add an additional Group named AnotherGroup to the previous example, you would specify:

  TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify one or more Users to be added separated by semicolons. The value of this field should be specified just like the Groups field:

  username;username;

  For Example, to associate a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User named Bob to the previous example, you would specify:

  Joe;Bob;

- *RemoteEmails* – (Optional) – The *RemoteEmails* field is used to specify one or more Remote Emails to be added separated by semicolons. The value of this field should be specified just like Groups and Users:

  remoteEmail;remoteEmail;

For example, to associate the Remote Email joe@somewhere.com you would specify:

joe@somewhere.com;

If you wanted to add an additional Remote bob@somewhere.com to the previous example, you would specify:

joe@somewhere.com;bob@somewhere.com;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.6. *Release Document*

Release Document allows you to release a Document.

Service Class: *wm.activity.ReleaseDocument*

Fields:

- *Revision* – (Mandatory) – The *Revision* field is used to specify the revision number to use for the released Document.
- *Render* – (Mandatory) – The *Render* field is used to specify whether or not the Document should be rendered. Use True or False.
- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

If the Document being released is a Metric Document, the following fields must also be specified:

- *MetricDate* – (Mandatory) – The *MetricDate* field is usually used to specify the reporting end date for a Metric. A Date in this field should be specified as:

  07/31/2022

- *MetricStatus* – (Mandatory) – The *MetricStatus* field is used to specify the metric status of the Document at the time of releasing. A *MetricStatus* field should be specified as:

  Green - Improving

  At the time of the writing of this guide, the valid Metric Status values are:

  Inactive
  Green - Improving
  Green - Staying Constant
  Green - Worsening
  Yellow - Improving
  Yellow - Staying Constant
  Yellow - Worsening
  Red - Improving
  Red - Staying Constant
  Red - Worsening

6.7. *Remove Access Association*

Remove Access Association allows you to remove access associations from a Document for multiple Groups, Users, and Remote Users in a single call. When removing access for a Group, User, or Remote User, only the name can be specified. When a Group, User or Remote User is specified for removal, all access associations for that Group, User, or Remote User are removed from the Document.

Service Class: *wm.activity.RemoveAccessAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify access to be removed for one or more Groups separated by semicolons. The value of this field should be specified as:

  groupName;

  For example, to remove a Group named *TestGroup*, you would specify:

  TestGroup;

  If you wanted to add an additional Group to remove named *AnotherGroup* to the previous example, you would specify:

TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify access to be removed for one or more Users separated by semicolons. The value of this field should be specified just like the Groups field:

  username;

  For Example, to remove a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User to remove named Bob to the previous example, you would specify:

  Joe;Bob;

- *RemoteUsers* – (Optional) – The *RemoteUsers* field is used to specify access to be removed for one or more Remote Users separated by semicolons. The value of this field should specified as:

  *authenticatorName**username;*

  For example, to remove a Remote User named Joe that authenticates using the Authenticator named *TestAuthenticator* you would specify:

  *TestAuthenticator**Joe;*

  Remote Users can also be specified as Remote User name @ Authenticator name. For example:

  joe@TestAuthenticator;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.8. *Remove Commenter Association*

Remove Commenter Association allows you to remove commenter associations from a Document for multiple Groups and Users in a single call.

Service Class: *wm.activity.RemoveCommenterAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify one or more Groups to be removed separated by semicolons. The value of this field should be specified as:

  groupName;groupName;

  For example, to remove a Group named *TestGroup* you would specify:

  TestGroup;

  If you wanted to add an additional Group to remove named *AnotherGroup* to the previous example, you would specify:

  TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify one or more Users to be removed separated by semicolons. The value of this field should be specified just like the Groups field:

  username;username;

  For Example, to remove a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User to remove named Bob to the previous example, you would specify:

  Joe;Bob;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.9. *Remove Distribution Association*

Remove Distribution Association allows you to remove distribution associations from a Document for multiple Groups, Users, and Remote Emails in a single call.

Service Class: *wm.activity.RemoveDistributionAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify one or more Groups to be removed separated by semicolons. The value of this field should be specified as:

  groupName;groupName;

  For example, to remove a Group named *TestGroup* you would specify:

  TestGroup;

  If you wanted to add an additional Group to remove named *AnotherGroup* to the previous example, you would specify:

  TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify one or more Users to be removed separated by semicolons. The value of this field should be specified just like the Groups field:

  username;username;

  For Example, to remove a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User to remove named Bob to the previous example, you would specify:

  Joe;Bob;

- *RemoteEmails* – (Optional) – The *RemoteEmails* field is used to specify one or more Remote Emails to be removed separated by semicolons. The value of this field should be specified just like Groups and Users:

  remoteEmail;remoteEmail;

For example, to remove the Remote Email joe@somewhere.com you would specify:

joe@somewhere.com;

If you wanted to add the additional Remote Email to remove bob@somewhere.com to the previous example, you would specify:

joe@somewhere.com;bob@somewhere.com;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.10. *Remove Keyword*

Remove Keyword allows you to remove a Keyword from a Document.

Service Class: *wm.activity.RemoveKeyword*

Fields:

- *KeywordName* – (Mandatory) – The *KeywordName* field is used to specify the name of the Keyword to remove.
- *KeywordValue* – (Optional) – The *KeywordValue* field is used to specify the value of the Keyword to remove. If the *KeywordValue* is not specified, all Keywords with the *KeywordName* specified will be removed from the Document.
- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.11. *Remove Notification Association*

Remove Notification Association allows you to remove notification associations from a Document for multiple Groups, Users, and Remote Emails in a single call.

Service Class: *wm.activity.RemoveNotificationAssociation*

Fields:

- *Groups* – (Optional) – The *Groups* field is used to specify one or more Groups to be removed separated by semicolons. The value of this field should be specified as:

  groupName;groupName;

  For example, to remove a Group named *TestGroup* you would specify:

  TestGroup;

  If you wanted to add an additional Group to remove named *AnotherGroup* to the previous example, you would specify:

  TestGroup;AnotherGroup;

- *Users* – (Optional) – The *Users* field is used to specify one or more Users to be removed separated by semicolons. The value of this field should be specified just like the Groups field:

  username;username;

  For Example, to remove a User named Joe, you would specify:

  Joe;

  If you wanted to add an additional User to remove named Bob to the previous example, you would specify:

  Joe;Bob;

- *RemoteEmails* – (Optional) – The *RemoteEmails* field is used to specify one or more Remote Emails to be removed separated by semicolons. The value of this field should be specified just like Groups and Users:

  remoteEmail;remoteEmail;

  For example, to remove the Remote Email joe@somewhere.com you would specify:

  joe@somewhere.com;

If you wanted to add the additional Remote Email to remove bob@somewhere.com to the previous example, you would specify:

joe@somewhere.com;bob@somewhere.com;

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.12. *Replace Keyword*

Replace Keyword allows you to replace the value of an existing Keyword on a Document.

Service Class: *wm.activity.ReplaceKeyword*

Fields:

- *KeywordName* – (Mandatory) – The *KeywordName* field is used to specify the name of the Keyword whose value should be replaced.
- *KeywordValue* – (Mandatory) – The *KeywordValue* field is used to specify the new value for the Keyword.
- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.13. *Reserve Document*

Reserve Document allows you to reserve a Document.

Service Class: *wm.activity.ReserveDocument*

Fields:

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.14. *Simple Web Request*

This operation gives the ability to be able to craft and send any kind of web request. This operation can be used to hand build anything from a simple HTTP get or basic a REST request to a large and complex SOAP operation call. Simple Web Request is also a great when unique web services need to be called that are propriety/one off and cannot be used by a standard Service Task because they do not follow a publicly accepted standard.

Service Class: *wm.activity.SimpleWebRequest*

Fields:

- *RequestUrl* – (Mandatory) – The request URL field must specify the web address where the request should be sent.
- *RequestMethod* – (Optional) – The HTTP request method i.e. GET, POST, PUT, etc. When a *RequestMethod* isn't specified, the request defaults to GET.
- *RequestHeaders* – (Optional) – One or more HTTP headers may be specified to be included in the request. Each header and value should be separated by a colon and if more than one header and header value is specified, they should be separated using a vertical bar. For example:

  Accept: text/plain|Accept-Charset: utf-8

- *RequestBody* – (Optional) – The body text to include in the request if a body is needed.

6.15. *Select Document*

Select Document is used to "Select" the Document that all subsequent operations will target when called. For example, if you select a Document with the Document number *testDocument*, all operations called afterwards would target *testDocument*. If you were to call Reserve Document, *testDocument* would be reserved.

When calling Select Document, you must either specify the Document number or Document ID of the Document to select.

Service Class: *wm.activity.SelectDocument*

Fields:

- *DocNumber* – The *DocNumber* field is used to select a Document by its Document number.

or

- *DocID* – The *DocID* field is used to select a Document but its Document ID.

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.16. *Send Email*

Send Email allows you to send an email from within a workflow Process.

Service Class: *wm.activity.SendEmail*

Fields:

- *To* – (Mandatory) – The *To* field is used to specify the email address where the email should be sent.

- *Subject* – (Optional) – The *Subject* field is used to specify the subject line of the email.

- *Body* – (Optional) – The *Body* field is used to specify the body of the email.

- *Cc* – (Optional) – The *Cc* field is used to specify the carbon copy recipients that should also receive this email.

- *Bcc* – (Optional) – The *Bcc* field is used to specify the blind carbon copy recipients that should also receive this email but not be visible to the other recipients.

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

It is important to note that Send Email does allow additional body sections (paragraphs) to be added to the email. The first paragraph is specified using the Body field. If more paragraphs are desired after just this first one, you may use the fields:

Body2, Body3, Body4, Body5, Body6, Body7, Body8, Body9, and Body10

to specify up to 9 additional paragraphs for the email.

6.17. *Unrelease Document*

Unrelease Document allows you to unrelease a Document.

Service Class: *wm.activity.UnreleaseDocument*

Fields:

- *Revision* – (Mandatory) – The *Revision* field is used to specify the revision number of the Document to unrelease.
- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

6.18. *Unreserve Document*

Unreserve Document allows you to unreserve a Document.

Service Class: *wm.activity.UnreserveDocument*

Fields:

- *Reason* – (Mandatory) – The *Reason* field must be specified when calling this operation. The value entered should be treated just as a Reason field in TechDoc explaining why you are performing this operation.

7. Script Task Helper Objects

The TechDoc BPMN Script Task can be used to get and set process variables as well as build and parse complex objects and blocks of data using JavaScript. Along with all of the basic JavaScript functions, TechDoc has many additional helper objects that are available to aid in building and parsing different types of data. In the sections below, we'll cover each of the helper objects and the functionalities they expose. Each of the helper objects are prefixed with DB (short for DocuBrain) to help with any ambiguity. It is advised that any process variable naming or JavaScript variable/object naming done on the user's part try and avoid anything starting with DB.

7.1. *DBBase64*

The *DBBase64* object can be used to encode and decode base 64 encoded data. The data can be sourced/saved directly from/to process variables. This object contains the following two functions:

- *decodeBytes* – This function takes a base 64 encoded string and return a byte array of decoded data.
- *encodeBytes* – This function takes a byte array of data and returns a base 64 encoded string.

As an example, one may base 64 encode a process variable using the following:

```
var encodedVal = DBBase64.encodeBytes(YOUR_PROCESS_VAR);
```

7.2. *DBHtmlParser*

The *DBHtmlParser* object is available to help with parsing HTML. Typically, this function is used after a Simple Web Request service task call to parse the response received. For example, a simple HTTP GET can be performed and then this function can be used to locate and save various pieces of information to process variables. To get started, simply create a new *DBHtmlParser* object passing the HTML to parse:

```
var htmlParser = new DBHtmlParser(YOUR_PROCESS_VAR);
```

Then you may get the root document element:

```
var docRoot = htmlParser.getRoot();
```

At this point, you now have a *JSoup* document root object, you can begin call any of the standard functions available in the *JSoup* library. For more information on *JSoup* and all that it has to offer, please visit https://jsoup.org.

7.3. *DBJsonBuilder*

The *DBJsonBuilder* object allows you to create a full JSON object without the need to construct the formatted string by hand. *DBJsonBuilder* takes a single parameter; a boolean that specifies whether or not the base JSON object is an array. If the base object should be created as an array, true should be specified. If a plain JSON object is needed, just specify false. For example, to create a plain JSON object one should enter the following:

```
var jsonBuilder = new DBJsonBuilder(false);
```

After this, one should begin by first getting the object root:

```
var root = jsonBuilder.getRoot();
```

Once the root is obtained, one can begin adding child fields and objects. For example, to add a few string keys and values field one could do the following:

```
root.add("title", "Title Here");
root.add("body", "Body Here");
root.add("userId", "42");
```

To modify the title field:

```
root.set(“title”, “new title here”);
```

To remove the title field all together:

```
root.remove(“title”);
```

Finally, child objects and arrays can be added using the same methods as listed above. For example, one could create a new child object called person and add it to the root.

```
// Create a new person object.
var personBuilder = new DBJsonBuilder(false);
var person = personBuilder.getRoot();
person.set(“name”, “John”);
person.set(“age”, 30);
```

```
// Add the person object to the root object.
root.add("person", person);

// To create an array of people, first create the individuals.
var person1Builder = new DBJsonBuilder(false);
var person1 = person1Builder.getRoot();
person1.set("name", "mike");
person1.set("age", 31);

var person2Builder = new DBJsonBuilder(false);
var person2 = person2Builder.getRoot();
person2.set("name", "Jim");
person2.set("age", 32);

// Now create an array to hold the people.
var peopleBuilder = new DBJsonBuilder(true);
var people = peopleBuilder.getRoot();
people.add(person1);
people.add(person2);

// Finally add the people array to the root object.
root.add("people", people);

// When it's time to save the JSON object to a string based process variable…
YOUR_PROCESS_VAR = root.toString();
```

For more information on the operators available, please visit:
https://github.com/ralfstx/minimal-json

7.4. *DBJsonParser*

The *DBJsonParser* object allows you to parse a JSON object without the need to walk a big string variable by hand. *DBJsonParser* takes a single parameter, the JSON string to parse. For example:

```
// Create a new instance of the JSON parser.
var jsonParser = new DBJsonParser(YOUR_PROCESS_VAR);

// Get the root JSON object to start working with.
var rootJsonObjOrArray = jsonParser.getRoot();

// Getting a string value.
```

```
var stringVal = rootJsonObjOrArray.get("myString");

// Getting child object.
var childObj = rootJsonObjOrArray.get("myChildObject");
```

For more information on the operators available, please visit:
https://github.com/ralfstx/minimal-json

7.5. *DBNow*

The *DBNow* object can be used to get the date and time at the moment it is called. This is handy as a workflow process might be built right now and not executed for months and you need the date and time at the very moment the process executes. *DBNow* can be used as follows:

```
// Get the current date and time in the ISO 8601 format.
var currentDateTime = DBNow.iso8601DateTime();

// Get the current date in day-month-year format.
var currentDate = DBNow.dateTime('dd-MM-yyyy');
```

The *dateTime* function accepts anything that Java allows using Java's *SimpleDateFormat*. For more information on the format, please visit:
https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

7.6. *DBSoapUtils*

The *DBSoapUtils* object can be when building a SOAP request to generate some of the various pieces needed to complete the request envelope. The object features the following functions:

```
// Generate a random message ID.
var msgID = DBSoapUtils.generateRandomMessageID();

// Generate random nonce bytes.
var nonceBytes = DBSoapUtils.generateRandomNonceBytes();

// Generate random nonce bytes in a base 64 encoded string.
var nonceBytes64 = DBSoapUtils.generateRandomNonceBase64();

// To generate a password digest using your nonce value (you may pass either
// nonceBytes or nonceBytes64), created (an ISO8601 date time), and a
```

```
// password.
var digest = DBSoapUtils.generatePasswordDigest(nonce, created, password);
```

This is all that exists now in the *DBSoapUtils* object but if there are any other core type functionalities you need to generate a SOAP message, let us know and we'll get them added.

7.7. *DBXmlBuilder*

The *DBXmlBuilder* object allows you to create a full XML object without the need to construct the formatted string by hand. For example, to create an XML object one should enter the following:

```
var xmlBuilder = new DBXmlBuilder();
```

After this, one should begin by first getting the object root:

```
var root = xmlBuilder.getRoot();
```

Once the root is obtained, one can begin constructing the document using all of the standard methods available in the Java XML API. For example, to create an element:

```
var elm = root.createElement('MyTagName');
```

To set an attribute on that element:

```
elm.setAttribute('attributeNameHere', 'attributeValueHere');
```

To remove an attribute:

```
elm.removeAttribute('attributeNameHere');
```

To remove a child element:

```
root.removeChild(elm);
```

For more information on the operators available, please visit:
http://docs.oracle.com/javase/6/docs/api/org/w3c/dom/Document.html

7.8. *DBXmlParser*

The *DBXmlParser* object allows you to parse an XML object without the need to walk a big string variable by hand. *DBXmlParser* takes a single parameter, the XML string to parse. For example:

```
// Create a new instance of the XML parser.
var xmlParser = new DBXmlParser(YOUR_PROCESS_VAR);

// Get the root XML object to start working with.
var root = xmlParser.getRoot();
```

There are many different functions available on the *DBXmlParser* instance.

7.8.1. containsAttribute

To check if an attribute exists on an element, the following function can be used:

```
if (xmlParser.containsAttribute(elementToCheck, attributeName))
```

Pass the element to check and the name of the attribute to look for. The function will return true if the attribute exists or false if it does not.

7.8.2. elementToString

To write and element to a string, the following function can be used. Specify the element to write out, true/false to skip the XML declaration, true/false to prettily format the text.

```
var str = xmlParser.elementToString(element, false, true);
```

7.8.3. findElementByID

This function can be used to traverse a single node, many nodes, or the entire document to locate and retrieve a single node by its ID. For example, to search the whole document:

```
var elementFound = xmlParser.findElementByID(root, 'ID_HERE');
```

7.8.4. findElementByName

This function can be used to traverse a node, node tree or document to locate and retrieve an element by its name. This function returns the first element it encounters. For example, to find and return the first element with the name 'color':

var element = xmlParser.findElementByName(nodeToSearch, 'color', true);

The last parameter specifies whether or not to ignore the namespace of the element. If true is specified, the namespace will be ignored and just *'color'* should be passed. If the namespace is important, pass false and specify *'yourNameSpace:color'*.

7.8.5. findElementsByAttributeValue

This function can be used to find and return all of the child elements under the element specified that have the attribute name and value specified. Additionally, the type of element should be specified i.e. 'div', 'input', etc. For example:

var divList = xmlParser.findElementsByAttributeValue(root, 'div', attributeName, attrbuteValue);

The return value of this function is an array.

7.8.6. getRoot

This function returns the root node of the document. For example:

var documentRoot = xmlParser.getRoot();

7.8.7. hideNodes

This function can be used to hide specific nodes from showing up in subsequent parse calls. For example, if you wanted to hide all of the div nodes that had a specific attribute name and value present you could:

xmlParser.hideNodes(root, 'div', 'favoriteColor', 'blue');

In this example, any *div* in the entire document that had an attribute named *favoriteColor* with a value of *blue* would be hidden.

7.8.8. parseAttributeValueAsBoolean

This function can be called to parse and return the specified attribute on an element as a boolean. For example:

> *var boolVal = xmlParser.parseAttributeValueAsBoolean(element, attributeName, defaultValue);*

The element and attribute name should be specified as well as the default value to return should the attribute not exist.

7.8.9. parseAttributeValueAsDouble

This function can be called to parse and return the specified attribute on an element as a double. For example:

> *var dblVal = xmlParser.parseAttributeValueAsDouble(element, attributeName, defaultValue);*

The element and attribute name should be specified as well as the default value to return should the attribute not exist.

7.8.10. parseAttributeValueAsInteger

This function can be called to parse and return the specified attribute on an element as an integer. For example:

> *var intVal = xmlParser.parseAttributeValueAsInteger(element, attributeName, defaultValue);*

The element and attribute name should be specified as well as the default value to return should the attribute not exist.

7.8.11. parseAttributeValueAsString

This function can be called to parse and return the specified attribute on an element as a string. For example:

> *var stringVal = xmlParser.parseAttributeValueAsString(element, attributeName, defaultValue);*

The element and attribute name should be specified as well as the default value to return should the attribute not exist.

7.8.12. parseAttributesByPrefix

This function can be called to parse and return all of the attributes of an element whose attribute name starts with the prefix specified. For example:

var attributeList = xmlParser.parseAttributesByPrefix(element, 'TechDoc');

In the example above, a list of attributes is returned that contains only attributes with names that start with TechDoc.

7.8.13. parseChildNode

This function can be called to parse and return a single child node of an element. For example:

var childNode = xmlParser.parseChildNode(parentNode, 'myTag', true);

In the example above, the first child node with a tag name of 'myTag' is returned. The third parameter specifies whether or not to ignore the namespace. If true is specified, just 'myTag' should be specified. If false is specified, 'myNamespace:myTag' should be specified.

7.8.14. parseChildNodes

This function can be called to parse and return multiple child nodes of an element. For example:

var childNodes = xmlParser.parseChildNode(parentNode, 'myTag', true);

In the example above, any child node with a tag name of 'myTag' is returned. The third parameter specifies whether or not to ignore the namespace. If true is specified, just 'myTag' should be specified. If false is specified, 'myNamespace:myTag' should be specified.

7.8.15. parseTextContent

This function can be called to parse and return the text content of the node specified. For example:

var textContentString = xmlParser.parseTextContent(node);

7.8.16. stripNamespace

This function can be used to strip the namespace off of the string specified. For example, if you performed a *parseChildNode* to select a particular element and wanted its tag name without the namespace you could:

var tagNameOnly = xmlParser.stripNamespace(myElement);

8. The TechDoc Server-Side Workflow Engine Reference

TechDoc features a complete BPMN 2.0 workflow engine and management system. The engine is tightly integrated with TechDoc and provides a large set of TechDoc operations in addition to all the generic features of BPMN. The management system provides both users and administrators with a management dashboard, real-time process monitoring tools, user task delegation and much more.

8.1. *Workflows Management Menu*

The Workflows Management menu contains all of the functioning for creating and managing Workflow Deployments, Definitions, Processes, and Process Triggers. This menu also displays Workflow Tasks that need attention, Tasks that are open to claim, and process instances started by you. Currently, this menu is only available to Users with the Workflows privilege or Administrators.

Navigation: *[DocMgr > Workflows]*

A User with the Workflows Manager or Admin privilege will have access to all of the Workflows Management functions. A User with the Workflows privilege will only have access to their Workflow Processes.

8.1.1. Creating a Workflow Deployment

Create Workflow Deployment creates a new Workflow Deployment in the Document Manager. A Workflow Deployment is a package (a .bar file) containing one or more Workflow Process Definitions their resources and/or images. In order to manually create a Workflow Deployment, you must save your process/processes from your BPMN editor as a .bar file (Activiti Business Archive File). Using the BPMN Editor, after creating a Workflow Process and saving it, select Save As from the File menu, and choose "Activiti business archive File (.bar)" from the Files of Type drop down when saving.

- The user must have the Workflows, Workflows Manager, or Admin privilege.

Navigation: *[DocMgr > Workflows > Side Menu > Create Deployment]*

Step 1:

1. Enter the name of the Deployment in the Name box. This is a required field. The maximum length of this field is 255 characters.
2. Select the Deployment owner in the Owner box by clicking on the down arrow and selecting a name from the list.

Note: Only an Admin can select the owner of a Deployment.

3. Enter reason for creating the Workflow Deployment in the Reason box. This is a required field. The maximum length of this field is 255 characters.
4. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

A Workflow Deployment file is an archive (.bar) file containing one or more Workflow Processes.

1. At the Deployment box click on the Browse... button to locate the Workflow Deployment to be load into the Document Manager.

   Note: The File Upload box will be displayed.

2. In the Look in box select the drive/folder where the Deployment to be stored in the Document Manager is located. To display all of the files in the folder, click on the down arrow and select All Files (*.*). Click on the Deployment to be stored in the Document Manager. This will automatically insert the filename in the File name box. Click the Open button.
3. Click the Cancel button to cancel the command, click the Previous button to go back to the previous screen, or click the OK button to create the Workflow Deployment.

Notes:

- A new Workflow Deployment Mapping record will be created.
- The Workflow Deployment will be inserted into the Workflow Engine.
- A history record will be generated for creation of the Workflow Deployment.

8.1.2. Deleting a Workflow Deployment

Delete Workflow Deployment deletes an existing Workflow Deployment in the Document Manager. Multiple steps are required during the process in order to minimize the chances of an accidental deletion.

- The User must have Owner access to the Workflow Deployment.

Navigation: *[DocMgr > Workflows > Side Menu > Deployments > Select Desired Deployment > Side Menu > Delete]*

Step 1:

The Deployment to be deleted and the Deployment attributes are displayed.

1. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

The Deployment to be deleted and the Deployment attributes are displayed.

1. Enter the reason for deleting the Deployment in the Reason box. This is a required field. The maximum length of this field is 255 characters.
2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the OK button to delete the Workflow Deployment.

Notes:

- The Workflow Deployment Mapping record will be deleted.
- The Workflow Deployment will be retracted from the Workflow engine.
- A history record will be generated for deletion of the Workflow Deployment.

8.1.3. Modifying a Workflow Deployment

Modify Workflow Deployment modifies an existing Deployment in the Document Manager. A Deployment is a bar file package that contains one or more workflow process definitions and various resource files needed by the deployment. Typically, Workflow Deployments are created using the DocuBrain Workflow Editor and are uploaded directly to the TechDoc Document Manager where they will execute. If they need to be modified, they should be downloaded using the Workflow Editor, modified, and then re-uploaded to the Document Manager. The Modify Workflow Deployment servlet only allows an Admin to re-assign the owner of the deployment.

- The user must have the Admin privilege.

Navigation: *[DocMgr > Workflows > Side Menu > Deployments > Select Desired Deployment > Side Menu > Modify]*

Step 1:

1. If applicable, modify the owner of the Workflow Deployment by clicking on the down arrow and selecting a new owner from the list.
2. Enter reason for modifying the Workflow Deployment in the Reason box. This is a required field. The maximum length of this field is 255 characters.
3. Click the Cancel button to cancel the command, or click the OK button to modify the Workflow Deployment.

Notes:

- The existing Workflow Deployment record will be modified.
- A history record will be generated for the modification of the Workflow Deployment.

8.1.4. Showing a Workflow Deployment

Displays Workflow Deployments created in the Document Manager

A Specific Workflow Deployment

Navigation: *[DocMgr > Workflows > Side Menu > Deployments > Select Desired Deployment]*

The User must have read access to the Workflow Deployment.

Field Name	Description
Deployment ID	The ID used by the Workflow engine to uniquely identify this Deployment.
Name	The name of this Workflow Deployment.
Created	The date and time the Deployment was created.
Deployment Time	The date and time the Deployment was deployed in the Workflow engine.
Owner	The User that owns this Deployment. Click the owner's name link to display the User Info screen.

Process Definitions

This section contains all of the Process Definitions contained in this Deployment. The Name, Process Definition ID, Key, and Version will be displayed for each Process Definition in the Deployment.

- Click on to Show Info of the specific Process Definition.

Deployment Resources

This section contains all of the resources contained in this Deployment. The filename will be displayed for each resource in the Deployment.

- Click on to Fetch the specific Deployment resource.

All Workflow Deployments
Navigation: *[DocMgr > Workflows > Side Menu > Deployments > Side Menu > All Deployments]*

All Workflow Deployments displays all the Deployments that have been created in the Document Manager. Note: Only a User with the Workflows Manager or Admin privilege can show All Deployments.

- The Deployment ID, Name, and Deployment Time are displayed for each Deployment.
- The number of Deployments is shown.
- Click on to Show Info for the specific Deployment.

My Workflow Deployments
Navigation: *[DocMgr > Workflows > Side Menu > Deployments > Side Menu > My Deployments]*

My Workflow Deployments displays all the Deployments that you currently own in the Document Manager.

- The Deployment ID, Name, and Deployment Time are displayed for each Deployment.
- The number of Deployments is shown.
- Click on to Show Info for the specific Deployment.

8.1.5. Showing a Workflow Process Definition

Displays Workflow Process Definitions created in the Document Manager

A Specific Workflow Process Definition

Navigation: *[DocMgr > Workflows > Side Menu > Process Definitions > Select Desired Process Definition]*

The User must have read access to the Workflow Process Definition.

Field Name	Description
Process Definition ID	The ID used by the Workflow engine to uniquely identify this Process Definition. This ID is a combination of the Process Definition's Key and Version number.
Name	The name of this Process Definition.

Version	The version of number of this Process Definition. "1" represents an original deployment of a Process Definition while subsequent numbers represent each revision.
Deployment ID	The ID used by the Workflow engine to uniquely identify the Deployment this Process Definition belongs to. Click the Deployment ID link to display the Deployment Info screen.
Deployment Name	The name of the Deployment this Process Definition belongs to.
Key	The Key of this Process Definition. The Key is the original id given to the process in the BPMN editor it was created in.
Created	The date and time the Process Definition was created.
Owner	The User that owns this Process Definition. Click the owner's name link to display the User Info screen.
Diagram Resource Name	The Deployment resource name of the thumbnail for this Process Definition.
Resource Name	The Deployment resource name of the BPMN XML file for this Process Definition.

Process Diagram

This is a thumbnail of the diagram representing the BPMN logic for this Process Definition.

All Workflow Process Definitions
Navigation: *[DocMgr > Workflows > Side Menu > Process Definitions > Side Menu > All Process Definitions]*

All Workflow Process Definitions displays all the Process Definitions that have been created in the Document Manager. Note: Only a User with the Workflows Manager or Admin privilege can show All Process Definitions.

- The Name, Process Definition ID, Key, and Version are displayed for each Process Definition.
- The number of Process Definitions is shown.
- Click on to Show Info for the specific Process Definition.

My Workflow Process Definitions
Navigation: *[DocMgr > Workflows > Side Menu > Process Definitions > Side Menu > My Process Definitions]*

My Workflow Process Definitions displays all the Process Definitions that you currently own in the Document Manager.

- The Name, Process Definition ID, Key, and Version are displayed for each Process Definition.
- The number of Process Definitions is shown.
- Click on to Show Info for the specific Process Definition.

Process Definitions in Deployment
Navigation: *[DocMgr > Workflows > Side Menu > Deployments > Select Desired Deployment > Side Menu > Process Definitions]*

Process Definitions in Deployment displays all the Process Definitions in the selected Deployment.

- The Name, Process Definition ID, Key, and Version are displayed for each Process Definition.
- The number of Process Definitions is shown.
- Click on to Show Info for the specific Process Definition.

8.1.6. Showing a Workflow Process Instance

Displays Workflow Process Instances in the Document Manager

A Specific Workflow Process Instance

Navigation: *[DocMgr > Workflows > Side Menu > Process Instances > Select Desired Process Instance]*

The User must have read access to the Workflow Process Instance.

Field Name	Description
Name	The Name of the Process Definition this Process Instance started from.
Process Instance ID	The ID used by the Workflow engine to uniquely identify this Process Instance.
Process Definition ID	The ID used by the Workflow engine to uniquely identify the Process Definition this Instance belongs to.
Ended	The date and time the Process Instance ended.
Suspended	Whether or not Process Instance is suspended.

Process Diagram

This is a thumbnail of the diagram representing the BPMN logic for this Process Instance. The activity circled in red indicates the currently running activity.

All Workflow Process Instances
Navigation: *[DocMgr > Workflows > Side Menu > Process Instances > Side Menu > All Process Instances]*

All Workflow Process Instances displays all the Process Instances in the Document Manager. Note: Only a User with the Workflows Manager or Admin privilege can show All Process Instances.

- The Name, Process Instance ID, Process Definition ID, Suspended, and Ended, values are displayed for each Process Instance.
- The number of Process Instances is shown.
- Click on to Show Info for the specific Process Instance.

All Workflow Process Instances that I Started
Navigation: *[DocMgr > Workflows > Side Menu > Process Instances > Side Menu > My Process Instances]*

All Workflow Process Instances that I Started displays all the Process Instances in the Document Manager that you started.

- The Name, Process Instance ID, Process Definition ID, Suspended, and Ended, values are displayed for each Process Instance.
- The number of Process Instances is shown.
- Click on to Show Info for the specific Process Instance.

Process Instances of Definition
Navigation: *[DocMgr > Workflows > Side Menu > Process Definitions > Select Desired Process Definition > Side Menu > Process Instances]*

Process Instances of Definition displays all the Process Instances in the Document Manager of the selected Process Definition.

- The Name, Process Instance ID, Process Definition ID, Suspended, and Ended, values are displayed for each Process Instance.
- The number of Process Instances is shown.
- Click on to Show Info for the specific Process Instance.

Showing a Workflow Process Instance Still Queue
Navigation: *[DocMgr > Workflows > Side Menu > Process Definitions > Select Desired Process Definition > Side Menu > Start Instance]*

Processes are not started directly within the workflow engine, they are queued. The workflow engine allows up to a set number of processes to run at one time as defined by the system property MaxWorkflowProcessInstances. When the max number of concurrent processes is exceeded, processes must wait in the queue before they begin. If there are available "slots" when a process is started/queued, it will begin immediately.

When a Process Instance is started manually and the queue is full, the Workflow Queued Process Info will be shown instead of the Process Instance Info. This screen can be manually refreshed periodically and once the Process begins, the running Process Instance Info will be shown instead.

- The ID, Created, Business Key, Message Name, Process Definition ID, Process Definition Key, Process Trigger ID, Retry Count and User values are displayed for each Process Instance.

8.1.7. Showing Workflow Element

Show Workflow Element provides a way to view the BPMN specific information for an individual element that is part of a Workflow process. The information displayed will vary based on the type of element being displayed.

- The User must have the Workflows privilege with Read access to the Workflow process.

Navigation: *[DocMgr > Workflows > Side Menu > Show Process Instances > Select Desired Process > Click on the Element to Display]*

Step 1:

The information shown for a Workflow element will vary depending on the type of element being displayed. All elements do have an ID, Name, and Documentation field, and most elements will have one or more incoming and outgoing sequence flows. The information displayed will directly correlate to the information displayed when viewing the element in the BPMN Editor. The ID shown is the ID that uniquely identifies an element in a Workflow process. Any reference to this element by other elements or connecting sequence flows will refer to this element using its ID.

8.1.8. Starting a Workflow Process Instance

Start Workflow Process Instance is used to queue a Process start in the Workflow Engine. A Process can be started from a Process Definition or started manually on a Document or Generation.

- The User must have read access to the Process Definition.

Starting a Process from Process Definition

Navigation: *[DocMgr > Workflows > Side Menu > Process Definitions > Select Desired Process Definition > Side Menu > Start Instance]*

When starting a Process Instance directly from a Process Definition, you will be redirected to either the running Process Instance or a screen showing the queued Process Instance. However, if the Process Definition requires you to complete a starting form, you must complete the form before the Process can start/queue. Starting forms are used in a Process to take in and set initial data for the Process. If a form is displayed, follow the steps below.

1. Follow any instructions displayed for this starting form.
2. Complete each field of the starting form that is required.
3. Click the Cancel button to cancel the command, or click the OK button to start the Process.

Notes:

- An instance of the Workflow Process will be queued to start on the Workflow Engine.

Starting a Process on a Document or Generation
Navigation: *[DocMgr > Explorer > Select Desired Document or Generation > Side Menu > Start Process]*

When starting a Process on a Document or Generation, the Document and/or Generation will be passed into the Process. All subsequent commands in the Process will target that Document and/or Generation.

- The User must have read access to the Document or Generation.

All of the available Processes to start are displayed.

1. Select a Process to start in the Process Trigger box by clicking the down arrow and choosing a Process Trigger from the list. The Process set in the Process Trigger selected will be the Process that is started. Only Process Triggers that have their Command set to "Start Process" can be started manually.
2. Enter the reason for starting the Process in the Reason box. Reason is a required field. The maximum length of this field is 255 characters.
3. Click the Cancel button to cancel the command, or click the OK button to start the Process.

Notes:

- An instance of the Workflow Process will be queued to start on the Workflow Engine.
- A history record will be generated for starting the Workflow Process.

8.1.9. Activating a Workflow Process Instance

Activate Workflow Process Instance is used to wake up a Process Instance that has been previously suspended by a User. This is sometimes useful because it allows you to stop and start a process. For example, if a Process Instance looks as though it might time out because of a timer task waiting on another condition (like a Document to be created) and you know the condition will be met shortly, you can suspend the Process Instance and then re-activate it afterwards so that you don't have to restart the entire Process over again.

8.1.10. Restart Workflow Process Instance Job

Restart Workflow Process Instance Job allows a job to be restarted after an error has occurred. After the process owner addresses the error condition, be it locating a document, adjust a process variable, etc., the user can restart the failed job/activity and allow the process to continue to completion.

- The user must have the Workflows, Workflows Manager or Admin privilege.
- The user must have Owner access to the process.

Navigation: *[DocMgr > Workflows > Side Menu > Show... > Process Instances > Select Desired Process Instance > Side Menu > Restart Job]*

Step 1:

A message is displayed that details the importance of understanding the job error and correcting the issue before any attempt to restart a job is made.

1. Click the Cancel button to cancel the command, click the Next button to continue.

Step 2:

The current process instance job errors are displayed in a dropdown box.

1. Select the error/job to retry.
2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the Next button to continue.

Step 3:

1. Click the Cancel button to cancel the command, click the Previous button to go back to the previous screen, or click the Next button to continue.

Step 4:

1. Enter the reason for restarting the failed job in the Reason box. This is a required field. The maximum length of this field is 255 characters.
2. Click the Cancel button to cancel the command, click the Previous button to go back to the previous screen, or click the OK button to perform the job restart.

Notes:

- The workflow process instance job will be retried.
- If the job fails again, a message will be displayed saying so and the process variables can be reviewed to see what the error states.
- A history record will be generated for the restarting of a failed workflow process instance job.

8.1.11. Suspending a Workflow Process Instance

Suspend Workflow Process Instance is used to pause the execution of a Process Instance. This is sometimes useful because it allows you to stop and start a process. For example, if a Process Instance looks as though it might time out because of a timer task waiting on another condition (like a Document to be created) and you know the condition will be met shortly, you can suspend the Process Instance and then re-activate it afterwards so that you don't have to restart the entire Process over again.

8.1.12. Deleting a Workflow Process Instance

Delete Workflow Process Instance deletes an existing Workflow Process Instance in the Document Manager. Multiple steps are required during the process in order to minimize the chances of an accidental deletion.

- The User must have Owner access to the Workflow Process Instance.

Navigation: *[DocMgr > Workflows > Side Menu > Process Instances > Select Desired Process Instance > Side Menu > Delete]*

Step 1:

The Process Instance to be deleted and the Process Instance attributes are displayed.

1. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

The Process Instance to be deleted and the Process Instance attributes are displayed.

1. Enter the reason for deleting the Process Instance in the Reason box. This is a required field. The maximum length of this field is 255 characters.
2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the OK button to delete the Workflow Process Instance.

Notes:

- The Workflow Process Instance will be deleted from the Workflow engine.
- A history record will be generated for deletion of the Workflow Process Instance.

8.1.13. Modifying Workflow Process Instance Variables

Modify Workflow Process Instance Variables allows the variables of a workflow process instance to be modified. Existing variables can be edited or removed and new variables can be added. This ability can be very handy for a process instance that has become stuck because of a business process change, design issue, etc.

Notes:

Before the variables of a workflow process instance can be modified, the instance must first be suspended.

After modifying the variables of a process instance, the instance will be activated again just before the variable changes are saved.

- The user must have the Workflows, Workflows Manager or Admin privilege.
- The user must have Owner access to the process.

Navigation: *[DocMgr > Workflows > Side Menu > Show... > Process Instances > Select Desired Process Instance > Side Menu > Modify Variables]*

Step 1:

A message is displayed that details the importance of understanding the process instance's design before any attempt at modification is made. Due to the complex nature of workflow, a single error in the adjustment, creation or removal of a variable can cause the entire process to terminate on error and become unrecoverable.

1. Click the Cancel button to cancel the command, click the Next button to continue.

Step 2:

The current process instance variables and their values are displayed. All the variables can be modified or removed with the exception of current user (if applicable). Like all other areas in TechDoc the owner or current user of an object can only be modified by an Admin and if a current user variable does exist on a process instance, it cannot be removed by anyone including an Admin. Additionally, if a current user variable does not exist on a process instance, one can be added. However, keep in mind once added it cannot be removed. New variables can be added by selecting a type from the drop down, entering a name for the variable next to the drop down and finally clicking the Add button.

1. Edit the values of any variables needed.
2. Remove any variables if needed.
3. Add any new variables needed.
4. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the Next button to continue.

Step 3:

1. Click the Cancel button to cancel the command, click the Previous button to go back to the previous screen, or click the Next button to continue.

Step 4:

1. Enter the reason for the modification of workflow process instance variables in the Reason box. This is a required field. The maximum length of this field is 255 characters.
2. Click the Cancel button to cancel the command, click the Previous button to go back to the previous screen, or click the OK button to perform the modification.

Notes:

- The workflow process instance will be activated.
- The variables changes will be made to the workflow process instance.
- Two history records will be generated. The first record will be created for the activation of the workflow process instance. The second record will detail all of the variable changes made on the workflow process instance.

8.1.14. Creating a Workflow Process Trigger

Create Workflow Process Trigger creates a new Process Trigger in the Document Manager. A Process Trigger is used to trigger the start of a selected Workflow Process after a command is executed and a preset condition is met by the specified Document attributes or Keywords. For example, if you were to create a Process Trigger that

specified a Workflow Process Definition named "MyFirstWorkflowProcess", the command "Create Document", and the Document Category "NS - Non-Sensitive Information", anytime a Document is created with the Document Category "NS - Non-Sensitive Information" a new instance of the "MyFirstWorkflowProcess" Process is started. When starting the instance, the Document that triggered the process becomes the "selected" document that all subsequent actions in the process will target (i.e. reserve document, release document, add keyword etc.). Therefore, when designing a Process that will be triggered by a Document, it is unnecessary to "select" the Document to work with in the Process as this will be done for you when the Process is triggered.

- The user must have the Workflows Manager or Admin privilege.

Navigation: *[DocMgr > Workflows > Side Menu > Create Process Trigger]*

Step 1:

1. Enter the Name for the Process Trigger in the Name box. The Process Trigger Name must be unique within the same Document Manager. The maximum length of this field is 255 characters.
2. Select the Process Trigger owner in the Owner box by clicking on the down arrow and selecting a name from the list.

   Note: Only an Admin can select the owner of a Process Trigger.

3. Select the command this Process Trigger should act upon in the Command box by clicking on the down arrow and selecting a command from the list.
4. Select a Process Definition in either the Process Definition ID or Process Definition Key drop down. You may select either one or the other, not both. A Process Definition has both an ID and Key. The ID is unique and can belong to only one version of one Process Definition whereas the Key is not unique and is used to identify all versions of a Process Definition. Each time a Process Definition is updated, a newer version is created. If you select a Process Definition by ID, it is guaranteed that that specific Process Definition will always be used even if a newer version is uploaded. If you select a Process Definition by Key, it is guaranteed that the latest version of a Process Definition will be used.
5. Enter reason for creating the Workflow Process Trigger in the Reason box. This is a required field. The maximum length of this field is 255 characters.
6. If the command is not "Start Process", additional criteria may be added to the trigger to further limit which documents will cause the trigger to fire. If an additional criterion is desired, select a Document attribute or Keyword from either the New Doc Attribute or New Doc Keyword box, click the Add button, then choose an operator and enter or select the value for the new criterion that was just added. The operator can have four values:

Operator	Description
Is	Tests if the value equals the document attribute or keyword after changes were made by the command.
Is Not	Tests if the value does not equal the document attribute or keyword after changes were made by the command.
Was	Tests if the value equals the document attribute or keyword before changes were made by the command. This operator can only be used with the Modify Document command.
Was Not	Tests if the value does not equal the document attribute or keyword before changes were made by the command. This operator can only be used with the Modify Document command.

7. "Start Process" does not allow any additional criteria. All other commands can optionally have one or more additional criteria.
8. Click the Cancel button to cancel the command, or click the OK button to create the Workflow Process Trigger.

Notes:

- A new Workflow Process Trigger record will be created.
- A history record will be generated for the creation of the Workflow Process Trigger.

8.1.15. Modifying a Workflow Process Trigger

Modify Workflow Process Trigger modifies an existing Process Trigger in the Document Manager. A Process Trigger is used to trigger the start of a selected Workflow Process after a command is executed and a preset condition is met by the specified Document attributes or Keywords. For example, if you were to create a Process Trigger that specified a Workflow Process Definition named "MyFirstWorkflowProcess", the command "Create Document", and the Document Category "NS - Non-Sensitive Information", anytime a Document is created with the Document Category "NS - Non-Sensitive Information" a new instance of the "MyFirstWorkflowProcess" Process is started. When starting the instance, the Document that triggered the process becomes the "selected" document that all subsequent actions in the process will target (i.e. reserve document, release document, add keyword etc.). Therefore, when designing a Process that will be triggered by a Document, it is unnecessary to "select" the Document to work with in the Process as this will be done for you when the Process is triggered.

- The user must have the Workflows Manager, or Admin privilege.

Navigation: *[DocMgr > Workflows > Side Menu > Process Triggers > Select Desired Process Trigger > Side Menu > Modify]*

Step 1:

1. If applicable, modify the Name for the Process Trigger in the Name box. The Process Trigger Name must be unique within the same Document Manager. The maximum length of this field is 255 characters.
2. Select the Process Trigger owner in the Owner box by clicking on the down arrow and selecting a name from the list.

   Note: Only an Admin can select the owner of a Process Trigger.

3. If applicable, modify the command this Process Trigger should act upon in the Command box by clicking on the down arrow and selecting a command from the list.
4. If applicable, modify the Process this Process Trigger should start by selecting a Process Definition in either the Process Definition ID or Process Definition Key drop down. You may select either one or the other, not both. A Process Definition has both an ID and Key. The ID is unique and can belong to only one version of one Process Definition whereas the Key is not unique and is used to identify all versions of a Process Definition. Each time a Process Definition is updated, a newer version is created. If you select a Process Definition by ID, it is guaranteed that that specific Process Definition will always be used even if a newer version is uploaded. If you select a Process Definition by Key, it is guaranteed that the latest version of a Process Definition will be used.
5. Enter reason for modifying the Workflow Process Trigger in the Reason box. This is a required field. The maximum length of this field is 255 characters.
6. If applicable, modify the Document attributes and/or Keywords used as additional criteria to determine which documents will cause this Process Trigger to fire. The attribute operator can have four values:

Operator	Description
Is	Tests if the value equals the document attribute or keyword after changes were made by the command.
Is Not	Tests if the value does not equal the document attribute or keyword after changes were made by the command.
Was	Tests if the value equals the document attribute or keyword before changes were made by the command. This operator can only be used with the Modify Document command.

Was Not	Tests if the value does not equal the document attribute or keyword before changes were made by the command. This operator can only be used with the Modify Document command.

7. "Start Process" does not allow any additional criteria. All other commands can optionally have one or more additional criteria.
8. Click the Cancel button to cancel the command, or click the OK button to modify the Workflow Process Trigger.

Notes:

- The existing Workflow Process Trigger record will be modified.
- The existing Workflow Process Trigger attribute records will be modified.
- A history record will be generated for the modification of the Workflow Process Trigger.

8.1.16. Deleting a Workflow Process Trigger

Delete Workflow Process Trigger deletes an existing Workflow Process Trigger in the Document Manager. Multiple steps are required during the process in order to minimize the chances of an accidental deletion.

- The User must have Owner access to the Workflow Process Trigger.

Navigation: *[DocMgr > Workflows > Side Menu > Process Triggers > Select Desired Process Trigger > Side Menu > Delete]*

Step 1:

The Process Trigger to be deleted and the Process Trigger attributes are displayed.

1. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

The Process Trigger to be deleted and any Process Trigger attributes are displayed.

1. Enter the reason for deleting the Process Trigger in the Reason box. This is a required field. The maximum length of this field is 255 characters.

2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the OK button to delete the Workflow Process Trigger.

Notes:

- The Workflow Process Trigger record will be deleted.
- Any additional Workflow Process Trigger attribute records will be deleted.
- A history record will be generated for deletion of the Workflow Process Trigger.

8.1.17. Showing a Workflow Process Trigger

Displays Workflow Process Triggers created in the Document Manager

A Specific Workflow Process Trigger

Navigation: *[DocMgr > Workflows > Side Menu > Process Triggers > Select Desired Process Trigger]*

The User must have read access to the Workflow Process Trigger.

Field Name	Description
Name	The name of this Process Trigger.
Process Definition ID	The ID used by the Workflow engine to uniquely identify the Process Definition of the Process this Trigger will start. When using an ID, it is guaranteed that that exact Process Definition will always be used. If a newer version is uploaded, this trigger will continue to use the older version you specified. This value will be empty if a Process Definition Key is used instead.
Process Definition Key	The Key used by the Workflow engine to identify the Process Definition of the Process this Trigger will start. When using a Key, the newest version of a Process Definition will be used. This means, after a trigger is created, every time a Process Definition is updated and a new version is created, this trigger will automatically begin use the new version in the future. This value will be empty if a Process Definition ID is used instead.
Owner	The User that owns this Process Trigger. Click the owner's name link to display the User Info screen.
Created	The date and time the Process Trigger was created.
Command	The command that this Process Trigger will act on.

Workflow Process Trigger Attributes

This section contains all of the attributes contained in this Process Trigger. The attribute Name, Operator, and Value will be displayed for each attribute in the Process Trigger.

All Workflow Process Triggers
Navigation: *[DocMgr > Workflows > Side Menu > Process Triggers > Side Menu > All Process Triggers]*

All Workflow Process Triggers displays all the Process Triggers that have been created in the Document Manager. Note: Only a User with the Workflows Manager or Admin privilege can show All Process Triggers.

- The Name and Process Definition ID are displayed for each Process Trigger.
- The number of Process Triggers is shown.
- Click on to Show Info for the specific Process Trigger.

My Workflow Process Triggers
Navigation: *[DocMgr > Workflows > Side Menu > Process Triggers > Side Menu > My Process Triggers]*

My Workflow Process Triggers displays all the Process Triggers that you currently own in the Document Manager.

- The Name and Process Definition ID are displayed for each Process Trigger.
- The number of Process Triggers is shown.
- Click on to Show Info for the specific Process Trigger.

8.1.18. Showing a Workflow Task

Displays Workflow Tasks in the Document Manager

A Specific Workflow Task

Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Select Desired Task]*

The User must have read access to the Workflow Task.

Field Name	Description
Task ID	The ID used by the Workflow engine to uniquely identify this Task.
Name	The name of this Task.

Description	The description of this Task. Typically, the description is a set of instructions on how to complete the Task.
Process Instance ID	The ID used by the Workflow engine to uniquely identify the Process Instance this Task belongs to.
Process Definition ID	The ID used by the Workflow engine to uniquely identify the Process Definition this Task belongs to.
Assignee	The User that is assigned to completed this task.
Owner	The User that owns this Task. Click the owner's name link to display the User Info screen.
Created	The date and time the Task was created.
Due Date	The date and time this Task must be completed.

All Workflow Tasks
Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > All Tasks]*

All Workflow Tasks displays all the Tasks in the Document Manager. Note: Only a User with the Workflows Manager or Admin privilege can show All Workflow Tasks.

- The Task ID, Task Name, and Description are displayed for each Task.
- The number of Tasks is shown.
- Click on to Show Info for the specific Task.

All Workflow Tasks that Need My Attention
Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > My Tasks]*

All Workflow Tasks that Need My Attention displays all Tasks you are assigned in the Document Manager that need your attention.

- The Task ID, Task Name, and Description are displayed for each Task.
- The number of Tasks is shown.
- Click on to Show Info for the specific Task.

All Workflows Tasks that are Open to Claim
Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > Open Tasks]*

All Workflows Tasks that are Open to Claim displays all Tasks in the Document Manager that are not yet assigned to a User and are open to claim.

- The Task ID, Task Name, and Description are displayed for each Task.

- The number of Tasks is shown.
- Click on to Show Info for the specific Task.

8.1.19. Completing a Workflow Task

Complete Workflow Task is used to complete a Task that you are assigned. Tasks are open-ended and can require you do just about anything before clicking to "complete" the Task. A Task can be a questionnaire you must complete, or be a set of instructions you must follow. A Task's instructions may require you to perform a task in another application, or manually perform an operation such as writing and sending an email, making a phone call to notify someone of something, physically take an envelope or package and mail it. A Task can instruct you to do just about anything.

- The User must the assignee on the Task.

Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > My Tasks > Select Desired Task > Side Menu > Complete]*

Step 1:

The Task to be completed is displayed. Read and follow any instructions displayed for the Task. Complete any form questions displayed. As soon as you are confident that you've completed the steps required to fulfill the request of the instructions, click the OK button.

1. Click the Cancel button to cancel the command, or click the OK button to complete this Task.

Notes:

- The Task will be completed and the Workflow Process will advance to the next activity.

8.1.20. Claiming a Workflow Task

Claim Workflow Task is used to claim an open Task that you are eligible to claim.

- The User must be a potential owner or be a member of a potential Group.

Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > Open Tasks > Select Desired Task > Side Menu > Claim]*

Step 1:

The Task to be claimed and the Task attributes are displayed.

1. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

The Task to be claimed and the Task attributes are displayed.

1. Enter the reason for claiming the Task in the Reason box. This is a required field. The maximum length of this field is 255 characters.
2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the OK button to claim the Workflow Task.

Notes:

- The Workflow Task will be assigned to you.

8.1.21. Assigning a Workflow Task

Assign Workflow Task is used to assign a Task to a User.

- The User must have the Workflows Manager or Admin privilege.

Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > All Tasks > Select Desired Task > Side Menu > Assign]*

Step 1:

The Task to be assigned is displayed.

1. Select an Assignee by clicking on the down arrow and selecting a name from the list.
2. Click the Cancel button to cancel the command, or click the OK button to assign this Task.

Notes:

- The Task will be assigned to the User selected.

8.1.22. Unassigning a Workflow Task

Unassign Workflow Task is used to unassign a Task.

- The User must have the Workflows Manager or Admin privilege.

Navigation: *[DocMgr > Workflows > Side Menu > Tasks > Side Menu > All Tasks > Select Desired Task > Side Menu > Unassign]*

Step 1:

The Task to be unassigned is displayed.

1. Click the Cancel button to cancel the command, or click the OK button to unassign this Task.

Notes:

- The Task will be unassigned.

8.1.23. Showing a Workflow Queued Process

Displays Workflow Queued Processes in the Document Manager

A Specific Workflow Queued Process

Navigation: *[DocMgr > Workflows > Side Menu > Queued Processes > Select Desired Queued Process]*

The User must have read access to the Workflow Queued Process.

Field Name	Description
ID	The ID used by the Workflow engine to uniquely identify this Queued Process.
Created	The date and time the Process was queued.
Business Key	If applicable, the business key of the Process Definition the Process Instance will start from.
Message Name	If applicable, the name of the start event message of the Process Definition(s) the Process Instance(s) will start from.
Process Definition ID	If applicable, the ID of the Process Definition the Process Instance will start from.
Process Definition Key	If applicable, the key of the Process Definition the Process Instance will start from.

Process Trigger ID	If applicable, the ID of the Process Trigger that will start the Process Instance.
Retry Count	Indicates the number of times the Queued Process has been retried to start a Process Instance. If this value is -1, the Queued Process has exceeded the maximum number of attempts and has been stalled. This typically happens with a Process Definition cannot be resolved because it no longer exists, the Process Trigger no longer exists or had an issue resolving a Process Definition, etc. If a stalled Queued Process is encountered, the Workflow Engine log file should detail the cause.
User	The owner of the Queued Process.

All Workflow Queued Processes
Navigation: *[DocMgr > Workflows > Side Menu > Queued Processes > Side Menu > All Queued Processes]*

All Workflow Queued Processes displays all the Queued Processes in the Document Manager. Note: Only a User with the Workflows Manager or Admin privilege can show All Queued Process.

- The ID, Create Date, Message Name, Process Definition ID, Process Definition Key, Process Trigger ID, and Retry Count, values are displayed for each Queued Process.
- The number of Queued Processes is shown.
- Click on to Show Info for the specific Queued Process.

All Workflow Queued Processes that I Started
Navigation: *[DocMgr > Workflows > Side Menu > Queued Processes > Side Menu > My Queued Processes]*

All Workflow Queued Processes that I Started displays all the Queued Processes in the Document Manager that you queued to start.

- The ID, Create Date, Message Name, Process Definition ID, Process Definition Key, Process Trigger ID, and Retry Count, values are displayed for each Queued Process.
- The number of Queued Processes is shown.
- Click on to Show Info for the specific Queued Process.

8.1.24. Purging All Stalled Workflow Queued Processes

Purge all stalled workflow queued processes purges all stalled workflow queued processes. Multiple steps are required during the process in order to minimize the chances of an accidental purge.

- The user must have the Admin or Workflow Manager privilege.

Workflow processes are not started directly but are queued to start when both the workflow engine is up and running and there is room for process instances to run. When the engine is stopped or at capacity, workflow processes remain in the queued state. The Workflow Scheduler background task wakes up periodically and starts processes from the queue when space is available in the engine. If a process should fail to start for any reason, it may become stalled. Once stalled, a message is logged in the workflow engine log. This log can be reviewed to correct the issue(s) for one or more stalled processes and then those stalled processes can be restarted. If for any reason the issue is not correctable or it's no longer needed for the process(es) to run, the stalled processes can be purged.

When stalled queued processes are purged, their records are removed from the database and corresponding processes will not be started for them.

Navigation: *[DocMgr > Workflows > Side Menu > Purge Stalled]*

Step 1:

1. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

1. Enter the reason for purging all of the stalled workflow queued processes in the Reason box. This is a required field.
2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the OK button to purge all of the stalled workflow queued processes.

Notes:

- All records in the Workflow Queued Process table that have reached the maximum number of retries will be deleted.

8.1.25. Restarting All Stalled Workflow Queued Processes

Restart stalled processes restarts all stalled Workflow Queued Processes. Multiple steps are required during the process in order to minimize the chances of an accidental restart.

- The user must have the Admin or Workflows Manager privilege.

Workflow processes are not started directly but are queued to start when both the workflow engine is up and running and there is room for process instances to run. When the engine is stopped or at capacity, workflow processes remain in the queued state. The Workflow Scheduler background task wakes up periodically and starts processes from the queue when space is available in the engine. If a process should fail to start for any reason, it may become stalled. Once stalled, a message is logged in the workflow engine log. This log can be reviewed to correct the issue(s) for one or more stalled processes and then those stalled processes can be restarted. If for any reason the issue is not correctable or it's no longer needed for the process(es) to run, the stalled processes can be purged.

When stalled queued processes are restarted, their retry count is set to zero and they will be eligible to start once again. The Workflow Scheduler will then attempt once again to start a process for each of those queued processes. This time, if no error occurs, the process will start normally. If any errors are encountered, each of the queued processes will become stalled once again.

Navigation: *[DocMgr > Admin > Restart Stalled]*

Step 1:

1. Click the Cancel button to cancel the command, or click the Next button to continue.

Step 2:

1. Enter the reason for restarting all of the stalled workflow queued processes in the Reason box. This is a required field.
2. Click the Cancel button to cancel the command, click the Previous button to return to the previous screen, or click the OK button to restart all of the stalled workflow queued processes.

Notes:

- All records in the Workflow Queued Process table that have reached the maximum number of retries will have their retry count field reset to zero.

9. Suggestions and Feedback

We hope you find the workflow engine to be very useful in automating processes in TechDoc. As the workflow engine is quite vast and very new, we have chosen to expose just the TechDoc operations outlined in this guide for this version. As we continue to build upon the workflow engine, we can begin to add more operations. If you have any suggestions about operations that should be added, how the workflow engine sections are laid out in TechDoc, etc., please let us know by sending us email at suggestions@docubrain.com

If you do happen to find a problem, please feel free to use the same email above. While we do prefer suggestions and compliments, we also accept criticism (preferably constructive :-)

At Prevo Technologies, Inc., we take the quality of our products very seriously. If you do find an issue or something that you feel can be improved upon in the TechDoc workflow engine, please let us know.