Concurrent Development

Updated on Dec 2, 2020 | 4 minutes to read

SBM Composer provides features that enable you and other designers to work on a process app at the same time.

> **Note:** This topic contains high-level information about concurrent development. For detailed information, see "Best Practices for Concurrent Development", which is available from the Composer Start Page.

Version Control

In SBM Composer, process apps contain design elements that represent the smallest unit of design. Design elements include custom forms, workflows, rules, and tables. Each design element is independent and has its own version. You must check out a design element before changing it, so no other designer can work with a design element until you check it in again. However, you and other designers can work on different design elements in the same process app simultaneously.

When you check out a design element, it is stored in the Local Cache on your computer. When you check in a design element, its version is incremented and it is stored in the repository, which is a shared location that you and other designers can access. The repository supports standard version control operations, as described in Repository Menu.

Before publishing a process app, note the following:

- If you do not use the **Get Latest** command before publishing a process app, the repository may have newer versions of design elements than those in your Local Cache. By default, when you publish, you are prompted to specify which versions should be used.
- When the **Get Latest** option is selected, the older versions in your Local Cache are overwritten by the newer versions before publishing takes place.

- When the **Ignore** option is selected, the older versions of the design elements are published. The newer versions in the repository are not overwritten. You might use this option when you want to deliver changes that you have made to a process app without having to uptake changes made by someone else that might interfere with your tests or are not necessary for you to uptake at this time.

For information about setting these options permanently, see Repository Options.

Compare and Merge

SBM Composer provides the ability to visually compare two versions of a process app and manually copy selected additions and changes from one process app to another. The typical use case for this feature is customers who upgrade from one version of a solution to another and want to keep the custom changes they made to the process apps in the earlier version. In this way, customers can combine aspects of their design with aspects of the solution developer's design.

> **Note:** For more information about this feature, see Compare and Merge.

Patch Context

You can use a "patch context" to patch a production environment while continuing to work on a process app in a development environment. This involves opening the labeled version of the process app that was deployed to production. When you work in a patch context, the repository commands such as **Check In**, **Check Out**, and **Get Latest** are applied against the patch context label.

> **Note:** For more information, see Working in a Patch Context.

Working in a Patch Context

Updated on Dec 2, 2020 | 5 minutes to read

A patch context allows parallel and concurrent development to occur on the same process app, so that patches can be applied to production process apps as ongoing development of the main process app continues. When performing maintenance work, a designer can create a patch context by opening a published process app by its label.

The patch context serves as the baseline for the maintenance work. Multiple process app designers can do concurrent development in a patch context.

Restriction: Only one process app designer can work on a single design element at the same time.

Changes a designer makes when working in a patch context apply only to the patch context. They do not affect ongoing development of the head (tip) version of the process app.

Note: A patch context is not the same as a branch in a version control application.

Some changes can be made directly in a patch context. If you want these changes to also apply to the head version, make them manually in the head version. Other changes cannot be made directly in a patch context, and are described in the following section.

Adding Design Elements to a Patch Context

To protect against data loss, the following design elements cannot be added directly to a patch context:

- Applications, including these design elements:

  - Tables
  - Fields
  - Selections for *Single Selection* and *Multi-Selection* fields
  - Application Workflows
  - States
  - Report Definitions
  - Roles

The following design Elements can be added to a Patch Context:

- Transitions
- Swimlanes
- Workflow Annotations
- Scripts (App, Mod, JS)

- State, Transition & Form Actions
- Forms
- State & Transition Privilege Over-rides
- Transition Field Over-rides

- Orchestrations and orchestration workflows

You can, however, copy these design elements from the head version to the patch version. If a design element does not exist in the head version, create it there first.

Restriction: Applications and orchestrations are exceptions to this. You cannot add applications or orchestrations to a patch context, and cannot copy them from the head version.

When you right-click the design element heading in App Explorer in the patch context, and then select **Add Existing <Design Element>**, the **Add Existing <Design Element> from the Head (Tip) Version** dialog box opens. In this dialog box, select the design element that you want to add and then click the **Add** button.

Note: Design elements containing fields, states, tables, and so on in the head version must be checked in before you can copy these entities to the patch context.

When adding items from the head version, the parent design element in the patch context must be checked out. For example, to add a workflow from the head version, make sure the application in the patch context is checked out.

After a table, application workflow, report definition, or role is added from the head version, that version of the design element is immediately labeled with the patch label, and the parent design element in the patch context is checked in (and checked out again if it was previously checked out).

After a field or state is added from the head version, the field or state is simply added to the parent design element and the parent design element in the patch context is checked in (and checked out again if it was previously checked out).

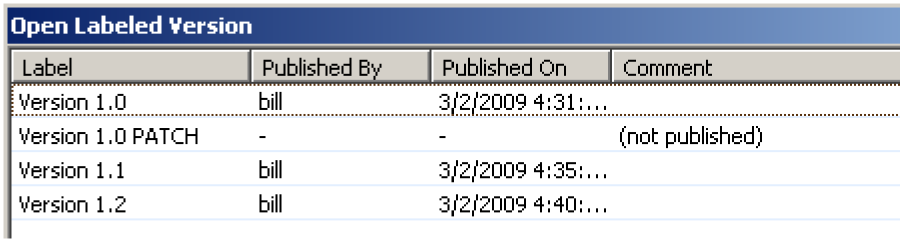
Example of Working in the Patch Context

Version 1.0 of a process app is in production. Susan checks out Version 1.0 (the head version) to continue main development on the process app.

At the same time, a patch needs to be applied to Version 1.0 for customers using the process app in production. The owner of a state needs to change. Bill opens Version 1.0 by its label in the **Open Labeled Version** dialog box. The *Version 1.0* label becomes *Version 1.0 PATCH*. Bill changes the owner of the state in the patch context (Version 1.0 PATCH).

Bill deploys Version 1.0 PATCH, and its label becomes *Version 1.1*. He manually changes the owner of the state in the head version so it matches the patch context. When he deploys the head version, its label becomes *Version 1.2*.

The **Open Labeled Version** dialog box resulting from this example is shown in the following illustration.



Open Labeled Version

Label	Published By	Published On	Comment
Version 1.0	bill	3/2/2009 4:31:...	
Version 1.0 PATCH	-	-	(not published)
Version 1.1	bill	3/2/2009 4:35:...	
Version 1.2	bill	3/2/2009 4:40:...	

Related Topics

Create Patch Context Dialog Box

Open Labeled Version Dialog Box

Add Existing Design Element from the Head (Tip) Version Dialog Box

Follow us on

VISIT SUPPORT

VISIT MARKETPLACE

SEND FEEDBACK

Add Existing Design Element from the Head (Tip) Version Dialog Box

Updated on Dec 2, 2020 | 26 seconds to read

Use this dialog box to copy a design element from the head (tip) version. This is the only way to add some design elements to a patch context.

For more information, see Working in a Patch Context.

Important: The design element you want to add must already exist in the head version.

Related Topics

Working in a Patch Context

Deploying Process Apps

Updated on Dec 2, 2020 | 4 minutes to read

Deployment is the act of taking a process app designed in SBM Composer and making it available on the SBM Application Engine server. You can configure your system so that deployments occur directly from SBM Composer, or you can deploy from the Process Apps tab in SBM Application Repository.

Endpoint and Target Server Considerations

Note the following important information:

- Although Application Repository automatically creates all system endpoints and also creates environment endpoints dynamically with each deployment, the environment endpoints require some user attention in case the "location hint" provided to Application Repository by SBM Composer does not match the SBM Application Engine endpoint.
- If you deploy a process app that contains a Web service, and the defining .wsdl file contains an incorrect service location for the environment to which you deployed, you need to edit the endpoint in Application Repository to point to the correct service location and then redeploy the process app so the changes can take effect. You should also get the latest changes back to SBM Composer.

  If you then clone the environment and deploy the same process app to the cloned environment using default settings, the new deployment will not use the existing endpoints that you edited; it will instead create new endpoints with similar names. These new endpoints will contain the service location from the defining .wsdl file, so you need to edit them (as you did in the original environment) to point to the correct service location for the Web service. (Alternatively, you can edit the deployment in the Deploy Options dialog box in SBM Composer to use the correct endpoints and delete the duplicate ones.) In either case, you need to redeploy the process app to the cloned environment so these changes can take effect.

Parallel Development Considerations

In a scenario in which multiple process app developers are working on process apps and there are multiple process app versions and patches, make sure that you do not undo someone's work that has already been deployed to the target environment (for example, as a patch to version 2.0 labeled 2.0.1) by deploying a process app version (for example, version 3.0) that does not contain that person's updates. If version 3.0 was developed directly from version 2.0 by User A, and meanwhile User B has created 2.0.1 as a deployed patch to version 2.0, the changes made by User B must be manually applied to the to the latest version before deployment.

The best approach is to always perform a "get latest" from the repository before deploying. This guarantees that you have the latest changes from other process app designers. If you don't do this, you risk undoing someone else's work. For example, a script might have been deployed by a co-worker and will now no longer be in production after your deployment.

If you want to make incremental changes to a process app currently running in an environment, the "patch context" feature enables you to do so. For more information, refer to the Composer help.

SBM Patch Context - How can we make a fix in Production whilst in the middle of new Development?

jgilmour

0 Likes

over 6 years ago

SBM Patch Context can often confuse SBM users, so I have put together a slide and this blog to explain. The slide may look busy but please look at it as a story and timeline, working from the top left across and then down through time.

Some fundamentals:

- When you Publish a Process App it get assigned auto-generated versions for design elements and then one for the actual whole Process App. This numbering is consecutive and system controlled. The highest version in main line is considered to be the TIP (or HEAD).
- At Publish time you also give a USER Defined LABEL.

In the real world the version of the Process App running in Development and Production is often different as ongoing development occurs.

The problem occurs when you need to make a change in Production but do not yet want the changes currently undergoing development to be include. This is where using Patch Context comes into play and you use the Open by Label to create a new branch of the Process App.

Important Considerations:

This is documented in the manual under the section "Adding Design Elements to a Patch Context". Certain elements like Workflows, States and Fields cannot be added within a Patch Context.

- Depending on the change you want to make in Production you may need to first make it in development to get the Asset UUID generated. You will not be able to make any changes in Patch Context that include a new asset with a UUID until you have first added to the TIP (also called HEAD). This ensure that when your development changes come through to Production your fix and associated date is not lost.
- If you have an asset checked out in the TIP then you cannot check it out in Patch Context. The converse is also true. You will get a "locked" messages. Please remember to check-in or undo checkouts when switching between the TIP and Patch Context.

Please now follow the story in the slide below and remember to start top left and across and then down (just like reading a book).