Best Practices for Concurrent Development with SBM
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who Should Read This Paper?</td>
<td>1</td>
</tr>
<tr>
<td>Version Control in SBM Composer</td>
<td>1</td>
</tr>
<tr>
<td>Compare and Merge in SBM Composer</td>
<td>7</td>
</tr>
<tr>
<td>Summary</td>
<td>8</td>
</tr>
</tbody>
</table>
Who Should Read This Paper?

This paper provides information about using Micro Focus® Solutions Business Manager (SBM) Composer features that enable concurrent development of process apps.

Target audiences of this paper include:

- Designers performing concurrent development using exclusive checkouts, technical contributors, and maintenance personnel (including Services, Support, Partner/Alliances), and architects trying to better understand version control for process apps. Information for these readers is found in Version Control in SBM Composer.

- Solution developers and customers wanting to merge custom changes they made to a process app in an earlier version of a solution into the same process app provided in a newer version of the solution. Information for these readers is found in Compare and Merge in SBM Composer.

Version Control in SBM Composer

The following sections provide information about how Solutions Business Manager stores design information, the version control features provided by SBM Composer, and best practices for using these features for doing development with multiple designers working on the same process app. In addition, they discuss features that support patching production systems while continuing to work on a development line.

- The Process App Blueprint and Design Elements

- What Is a Design Element?

- Design Element Storage

- Process App Development Use Cases for Version Control

- Patching Production Systems While Continuing New Development

The Process App Blueprint and Design Elements

The process app blueprint is the basic unit of development in SBM and represents a complete design of one or more applications and orchestrations. Process app blueprints are created using SBM Composer, published to Application Repository, and finally deployed to the SBM Application Engine and the SBM Orchestration Engine. A process app blueprint is a single file containing information fully describing any applications and orchestrations contained in it.

While this single file format is appropriate for transporting the design of a process app between components, the use of a single file does not facilitate concurrent development using exclusive checkouts. Because of this, process apps being designed in SBM Composer are comprised of a number of separate versioned and independent design elements that represent the smallest unit of the independently versionable process app design. For example, one developer can check out and work on a custom form while another developer designs an orchestration workflow.
What Is a Design Element?
A design element is the smallest part of a process app that can be independently checked in or out. It is loaded from and stored to a single corresponding file on the designer’s local disk. Examples of design elements include the following:

- The process app (without the applications and orchestrations it contains)
- Each application (without the contained parts)
  - Each workflow defined in the application
  - Each form with constituent controls defined in the application
  - Each image, icon, role, script, table, and Web service defined in the application
  - The application’s styles
  - The application’s type library
- Each orchestration (without the contained parts)
  - Each process app tool and orchestration workflow defined in the orchestration
  - Each Web service imported into the orchestration

In other words, every icon in the Process App Explorer represents either a design element or a collection of design elements that belong to a parent design element.

These design elements can be individually checked in or out. When adding or removing a design element, you need to check out the parent design element that references it. For example, if you are adding an application to a process app, you will need to check out the process app. Similarly, if you are adding or deleting a form or other design element under an application, you will need to check out that application (but not the process app).

Design elements can exist in any of three states. If it is new (it has never been checked in), it is editable by SBM Composer without restriction. If it has ever been checked in, it is either in a checked out state (editable) or a checked in state (not editable). New items have a ‘+’ overlay on their icons in the Process App Explorer, checked out items have a ‘✔’ overlay, and checked in items have a lock overlay.

Design Element Storage
The design elements that comprise a process app can be stored on the user’s local disk, in the repository version control system, or in both locations.

THE LOCAL CACHE
The design elements in a process app are stored in the local cache on the machine where SBM Composer is installed. By default this local cache is located in the current user’s Local Settings\Application Data directory and stores the design elements of all process apps that are new or have been retrieved from the repository.

NOTE: On Windows 7, the location is C:\Users\<username>\AppData\Local\Serena\Studio

For SBM Composer to open a process app for display or editing, the constituent design elements must exist in the local cache.
THE REPOSITORY
The repository is a shared location that multiple developers can connect to for collaboration in the design and development of a process app. Design elements are checked in and out of the repository. A check-in creates a new version of the design elements and locks the copy of the design element in the local cache so that it cannot be modified without checking it out for modification. This pessimistic locking (or exclusive checkouts) prevents designers from simultaneously making changes to the same design element.

The repository supports standard version control operations such as:
- Get Latest
- Check Out
- Check In
- Undo Check Out
- Version History

If someone has checked out a design element and is no longer available to either undo the check out or check in the changes, a designer with the Break Lock privilege in Application Repository can resolve the problem. When this designer opens a process app from the repository, or performs a get latest operation, the constituent design parts are copied to the local cache.

The user can only perform version control operations when SBM Composer is connected to the repository. Users not connected to the repository are said to be working offline and cannot modify design elements that they have not checked out; however, they can create new process apps and edit design elements that they checked out when they were last connected.

WHAT DESIGN IS PUBLISHED AND DEPLOYED?
When a process app is published (and subsequently deployed) from SBM Composer, it contains the content of the process app as it appeared in SBM Composer at the time of the publish and/or deploy operations. If a designer has not gotten latest, the published process app might contain design element versions older than the tip.

For example, suppose two designers are working on the same process app. One is working on form design and the other is working on an orchestration. The process app was copied into the local cache of each designer when it was first opened, or when they used the get latest command. Suppose the form designer has checked in changes to the form. If the orchestration designer has not gotten latest of the form design element, every deployment he makes will contain the old version of the form, because that is the version that exists in his local cache.

There are a number of benefits to basing the published content on the versions of the design elements in the local cache. A designer can work with versions of design elements that are known to be good by getting latest when the process app is in a stable state. The designer is then free to make modifications to only that part of the process app he is working on without being subject to the risk that unrelated changes might break validation or deployment. The patch context feature described below also depends on this capability, since the patch context collects earlier labeled versions of the design elements in the local cache. When patching a system, these earlier versions are used for the deployed content.
This behavior is by design and is analogous to the behavior of source control systems used by program-mers. The desired content is assembled locally before the resulting application or system is built and deployed.

**Process App Development Use Cases for Version Control**

SBM Composer can be used to develop new process apps, create new versions of existing process apps, and patch production systems while development work on new versions is ongoing. Work on a process app may be done by individuals or by teams. The sections below describe features of SBM Composer that apply to these circumstances.

**DESIGNERS WORKING ALONGE**

Many designers work on a single process app by themselves without collaboration. For these designers, it is often simplest to keep all design elements in the process app checked out and editable at all times. To support these designers, SBM Composer has the ‘check out all’ and ‘check in all’ commands. The ‘keep design elements checked out’ deployment option permits the design elements to be versioned (i.e., checked in) without changing their checked-out state.

**DESIGNERS WORKING ON DIFFERENT DESIGN ELEMENTS**

In some environments, teams of designers with specialized skills will work on independent portions of a process app. For example, one person might be an expert in developing orchestration workflows, while another might have expertise in form design. In this case, each designer will get latest to obtain the tip version of all design elements, and then check out only those design elements they need to work on.

Occasionally, there might be conflicts where two designers need the right to modify the same design element. For example, if a form design specialist needs to add a new form to an application and another designer needs to add a new auxiliary table to the application, they cannot do so simultaneously. The pessimistic locking mechanism will permit the first designer who attempts it to check out the parent application and the second will be prevented from doing so. To avoid conflicts like these, designers should be sure to only keep checked out those design elements that they are actively working on. In the example, the designer who has added a form to the application should check in both the form and the application to the repository as soon as the form is added and then check out the form for additional customization. This will enable the other designer to check out the application for adding the auxiliary table.

**DESIGNERS WORKING ON THE SAME DESIGN ELEMENTS**

SBM Composer does not support different designers working on the same design element at the same time from different instances of SBM Composer. When one designer checks out a design element, others are prevented from making changes to it. This is true even when one designer is working in a patch context on an earlier version of the design part and another is working against the development tip. In this case, the work on the design element must be completed by one designer before the other can begin work.

**Patching Production Systems While Continuing New Development**

SBM Composer supports patching a production system while continuing the development of a process app, through the use of a patch context.
CREATING A PATCH CONTEXT
To create a patch context, choose the Open command from the menu, select the Look In: repository radio button, select the process app of interest, and then click the Open labeled version button at the bottom of the dialog. The Open Labeled Version dialog appears, containing a list of all existing versions of the process app. Select the version that was deployed to the production system and click the OK button.

A dialog appears offering to create a patch context based on the selected label. After clicking OK, you are prompted to specify a label for the patch context. The label you choose will now become part of the version history for this process app and SBM Composer will open the patch context version of the process app.

The title bar of SBM Composer will always display the version label and distinctive text when you are operating in a patch context. For example, the title could read:

Hardware and Software Management (Version 1.3 PATCH) [Patch]

In this case, the label for the patch context is “Version 1.3 PATCH” and the “(Patch)” text indicates that the process app has been opened in a patch context.

OPENING AN EXISTING PATCH CONTEXT
If you have previously created a patch context for a process app, you can reopen that patch context at any time. When a patch context is created, a new version label is created corresponding to it. To open it, click the Open labeled version button in the Open Process App dialog as above, but when you are presented with the list of versions to open, open the version labeled with the patch label.

If the last version of a process app in the local cache was a patch context, you can open it by clicking its link in the recent process apps list on the SBM Composer menu. In this case, the label for the patch context will appear in parentheses following the process app name. For example, the link might read:

Hardware and Software Management (Version 1.3 PATCH) [Patch]

In 2009 R1 you should not open a patch context from the local cache using the Open dialog because it could lead to unexpected consequences. This limitation was addressed in 2009 R2.

MAKING CHANGES IN A PATCH CONTEXT
After you have opened a process app in a patch context, you can check out and modify the design elements that were used to create the patch context version of the process app.

Note: If a design element is checked out on the development tip, or in another patch context, it cannot be checked out again on the tip or any label. So if you need to make changes to a design element, you must make sure that anyone currently editing it checks it in before you attempt the change in the patch context.

Many changes can be made in the same way that they would if you were building a new process app, but others cannot. For example, when you are in a patch context, the workflow palette for application workflows does not contain any states. Instead, it contains the text ‘new states cannot be added because you are currently working in a patch context.’ The following sections describe the problem and the solution provided by SBM Composer in SBM 2009 R1 and later.
THE PROBLEM
If a designer adds a design component that affects the structure of the database in the SBM Application Engine, items created might contain references to that component. Consider the following scenario.

A designer patches a production system to add a new state. The application is run for a while generating items, some of which are in this state, some of which have this state as part of their history. He then deploys a newer version of the process app which, because of an oversight, doesn't contain this state.

In this scenario, records in the added state during the patch will no longer be visible, since the state was deleted by the second deployment.

Since every object in a process app is uniquely identified by a UUID (even if the designer were to independently create the new state in both the development and production versions of the process app), these states would actually have different identities, which would cause the same problem described above.

THE SOLUTION
To maintain consistency in the system and to prevent data loss, we must ensure that any new design component added to the patch context also exists in the development tip. This assures that when the development system is ultimately deployed, the design component added to the patch context will still exist.

We do this by prohibiting those design components that have a runtime presence (i.e., to which this requirement applies) from being created directly in a patch context. For example, the following items cannot be created new when operating in a patch context:

- Tables
- Fields
- Selections
- Application Workflows
- States
- Report Definitions
- Roles
- Applications
- Orchestrations

However, in place of creating such a design component, you can perform a ‘copy from tip’ operation. For example, to add a new state to your production environment, you can add it to the version control tip in development, open a patch context, and then perform a ‘copy from tip’ to copy it into the patch context.

In practice, the designer might realize the need for the new state while operating in the patch context. In this case, the designer would just open the tip version of the process app, make the change there, and then open the patch context again and copy the state from the tip.

DEPLOYING THE PROCESS APP TO PRODUCTION
After you have assembled the patched process app in SBM Composer, you can deploy it to whatever environment you want. If you have a production staging system, you might choose to deploy it there for
testing before deploying to production. If you will be testing it in your development environment, it will overwrite the current version of the process app there, so you might need to coordinate this with any testers or others expecting to see the process app version under development. The deployed process app will be comprised of the design elements you have in SBM Composer at the time you publish and deploy the process app.

**MAKING ADDITIONAL CHANGES TO PRODUCTION**

After the production version of a process app has been patched, it is even easier to do it a second time, since the patch context label already exists. Simply open the patch context label you previously made for the runtime environment and begin editing.

**Compare and Merge in SBM Composer**

SBM Composer provides the ability to visually compare two versions of a process app and manually copy selected additions and modifications from one process app to the other. This is particularly useful for customers who use SBM solutions.

Customers typically customize the process apps in a solution to meet their specific needs and processes. The process apps are often updated in a new version of a solution. This can be a problem for customers who want the improvements that the new process app offers, but do not want to lose their custom changes.

The compare and merge feature solves this problem. A customer can do the following in SBM Composer:

1. Run a report that identifies the custom changes that were made to the original process app.
2. Open the new and custom version of the process app side-by-side and see the differences clearly marked.
3. Use the report to locate and copy items that were added to the custom process app to the new process app, and modify items in the new process app that were modified in the custom process app.

Changes can be copied to either version of the process app. For example:

- A company has version 2.1 of a solution. Paul made modifications to one of the process apps. The company wants to upgrade to version 3.0 to adopt new features and enhancements, but wants to keep the changes Paul made. Paul uses the compare and merge feature in SBM Composer to examine the differences between the version 2.1 and 3.0 process apps. Because the changes made in version 3.0 are quite substantial compared to his changes, he copies his changes from version 2.1 into version 3.0.

- After version 3.0 was deployed, Ramani made additional changes to the process app. The company now wants to upgrade to version 3.1 to adopt bug fixes. The version 3.1 changes are minor, so Ramani copies the modifications from version 3.1 to the version 3.0 process app that she customized.
Summary

SBM Composer supports non-branching pessimistic locking version control that permits concurrent development using exclusive checkouts by designers working on different parts of the same process app. Design elements are locked when checked out so that only one designer may work on a design element at a time.

Patching of a production system is supported through the use of patch contexts. When a process app is published, all its constituent parts are labeled to identify what was used to create the process app. A patch context is created by opening a process app using a label, which delivers the design elements associated with that label to the local cache of the designer’s machine. These design elements can be checked out and edited to a patched process app. After the patch has been created, it can be deployed to any desired environment for testing.

SBM Composer also supports visually comparing and merging changes between two versions of a process app. The typical use case for this feature is customers who upgrade from one version of a solution to another and want to keep custom changes that they made to process apps in the earlier version.

Learn More At
www.microfocus.com/serena/support
www.microfocus.com/serena/